New tools and specification languages for biophysically detailed neuronal network modelling

Patrick John Gleeson

Department of Neuroscience, Physiology and Pharmacology

UCL

February 2012

This is a thesis submitted to the University of London in the Faculty of Life Sciences for the degree of Doctor of Philosophy. I confirm that the work presented is my own. Where information has been derived from other sources it has been indicated.

Abstract

Increasingly detailed data are being gathered on the molecular, electrical and anatomical properties of neuronal systems both *in vitro* and *in vivo*. These range from the kinetic properties and distribution of ion channels, synaptic plasticity mechanisms, electrical activity in neurons, and detailed anatomical connectivity within neuronal microcircuits from connectomics data. Publications describing these experimental results often set them in the context of higher level network behaviour. Biophysically detailed computational modelling provides a framework for consolidating these data, for quantifying the assumptions about underlying biological mechanisms, and for ensuring consistency in the explanation of the phenomena across scales.

Such multiscale biophysically detailed models are not currently in widespread use by the experimental neuroscience community however. Reasons for this include the relative inaccessibility of software for creating these models, the range of specialised scripting languages used by the available simulators, and the difficulty in creating and managing large scale network simulations.

This thesis describes new solutions to facilitate the creation, simulation, analysis and reuse of biophysically detailed neuronal models. The graphical application neuroConstruct allows detailed cell and network models to be built in 3D, and run on multiple simulation platforms without detailed programming knowledge. NeuroML is a simulator independent language for describing models containing detailed neuronal morphologies, ion channels, synapses, and 3D network connectivity. New solutions have also been developed for creating and analysing network models at much closer to biological scale on high performance computing platforms. A number of detailed neo-

cortical, cerebellar and hippocampal models have been converted for use with these tools.

The tools and models I have developed have already started to be used for original scientific research. It is hoped that this work will lead to a more solid foundation for creating, validating, simulating and sharing ever more realistic models of neurons and networks.

Contents

List of Figures

List of Tables

List of Publications

Journal Publications

Crook S, Gleeson P, Howell F, Svitak J, Silver RA (2007) MorphML: Level 1 of the NeuroML standards for neuronal morphology data and model specification. *Neuroinformatics* 5(2):96104

Cannon R, Gewaltig MO, Gleeson P, Bhalla U, Cornelis H, Hines M, Howell F, Muller E, Stiles J, Wils S, De Schutter E (2007) Interoperability of neuroscience modeling software: Current status and future directions. *Neuroinformatics* 5(2):127-138

Gleeson P, Steuber V, Silver RA (2007) neuroConstruct: a tool for modeling networks of neurons in 3D space. *Neuron* 54(2):219-235

Gleeson P, Crook S, Cannon RC, Hines ML, Billings GO, Farinella M, Morse TM, Davison AP, Ray S, Bhalla US, Barnes SR, Dimitrova YD, Silver RA (2010) NeuroML: A language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Computational Biology* 6(6):e1000, 815

Vervaeke K, Lőrincz A, Gleeson P, Farinella M, Nusser Z, Silver RA (2010) Rapid desynchronization of an electrically coupled interneuron network with sparse excitatory synaptic input. *Neuron* 67(3):435451

Book Chapters

Gleeson P, Silver RA, Steuber V (2010) Computer Simulation Environments. In: Cutsuridis V, Graham B, Cobb S, Vida I (eds) *Hippocampal Microcircuits: A Computational Modeler's Resource Book*, Springer Series in Computational Neuroscience, vol 5, Springer New York, pp 593-609

Gleeson P, Steuber V, Silver RA (2008) Using neuroConstruct to develop and modify biologically detailed 3D neuronal network models in health and disease. In: Soltesz I, Staley K (eds) *Computational Neuroscience in Epilepsy*, Academic Press, San Diego, pp 48-70

Gleeson P, Steuber V, Silver RA, Crook S (in press) NeuroML. In: Le Novère N, Bhalla US (eds) *Computational Systems Neurobiology*, Springer

Acknowledgements

I would like to thank my supervisor Professor Angus Silver for guiding me through this process and for teaching me the importance of communicating ideas clearly. His emphasis on making the computational resources I have developed relevant for and accessible to the wider neuroscience community has been crucial in their successful uptake.

I am also very grateful to the current and past members Silver Lab. It has been an excellent environment for developing the tools and models described in this thesis, providing critical feedback from experimental, theoretical and computational points of view.

I would also like to thank the users of neuroConstruct and the wider NeuroML community, especially Sharon Crook and Robert Cannon, for valuable contributions and feedback.

I thank my parents and family for all of their encouragement over the past number of years. Finally, I would like to thank my wife Anissa for her love and support during the long process of writing this thesis.

This work was supported by an MRC Special Research Training Fellowship in Bioinformatics and a Wellcome Trust Biomedical Resources Grant.

Chapter 1

Introduction

1.1 Computational modelling in scientific research

The use of mathematical models in the physical sciences has a long history. Consolidating knowledge of a physical system into a set of equations which can be used to make predictions about its behaviour under specific conditions is one of the core activities in science. Physics has been particularly successful at creating succinct mathematical descriptions of natural phenomena which provide insight into the underlying principles of the world around us. When a system can be described by a small number of interacting entities and a model created with a handful of variables, it is often possible to arrive at an analytical solution for the behaviour of the model (e.g. position in time of a planet moving around the sun or the rate of radioactive decay of atomic nuclei).

In many cases however, the complexity of the system under study does not allow a simple solution for how the state of the system will change with time, or behave under certain perturbations. While each element of the system may obey simple physical laws, how these interact to produce large scale system behaviour is not immediately obvious due to the nonlinear interaction of these elements (Gell-Mann, 1995). In recent years however, the use of computational modelling has greatly facilitated investigation of the behaviour of such systems. Software representations of the equations describing the model can be created and multiple simulations run under different input conditions or with different stochastic behaviour. Many fields have benefited from these advances (e.g. climate science, astrophysics, particle physics, structural chemistry, epidemiology) and they have enabled researchers to make predictions about the activity of the physical systems under different conditions and gain insight into their underlying organisational principles.

Computational modelling has become a core activity in physics, engineering and chemistry and biologists are also starting to benefit from encoding knowledge of a biological system in a form amenable to use in simulations. The data deluge associated with genomics and DNA sequencing has led to a range of new computational techniques being used in the field, under the umbrella of bioinformatics. New researchers with computational backgrounds have entered biology and nowadays most biologists routinely use software packages to acquire, manage and analyse their data.

A large part of the computational modelling in biology (above the scale of structural molecular biology) is related to encoding knowledge about biochemical signalling pathways, which has led to the emergence of the field of Systems Biology (Kitano, 2002). There is also increasing work at the level of cellular interactions and indeed whole tissue modelling (Noble, 2002).

Neuroscientists too are increasingly encoding what is known about the anatomy and physiology of the nervous system at different biological scales in computer models. Use of these models by experimental neurobiologists has been slow however, despite over a hundred years of constructing mathematical models of neurons and networks (e.g. Lapicque's 1907 work leading to the Integrate and Fire model (Brunel and van Rossum, 2007), McCulloch-

Pitts artificial neural networks (McCulloch and Pitts, 1943), Hodgkin Huxley squid axon model (Hodgkin and Huxley, 1952), Rall's work on cable theory (Rall, 1959)). The work in this area has much to gain from experiences in other fields, not only technically (e.g. simulation algorithms, use of high performance computing), but also from the social point of view: how models and software are shared, how standards are developed and how modelling results are communicated to experimentalists in the field.

1.2 Biophysically detailed neuronal modelling

Neuroscience has a number of characteristics which distinguish it from many other areas of investigation in the physical sciences and make computational modelling a vital tool for a greater understanding of the systems involved. Firstly, the elements which are involved are fundamentally multiscale: explaining the behaviour of a microcircuit, neuron, active membrane or synapse will inevitably involve assumptions about the behaviour of an entity on a lower biological scale. Secondly, signalling and computation are distributed in space and time, and it is not usually apparent whether subcellular signalling, synaptic strengths, individual cell or population activity (or a combination of these) is the fundamental substrate of an information processing task. Thirdly, many neural systems consist of numerous repeated instances of subelements (channels, neurons, columns, etc). Computer models are able to encode these hierarchical descriptions and easily make multiple repeated instances of subelements, in order to simulate the behaviour of the larger system under varying conditions. Finally, and most importantly, the elements of the nervous system interact in nonlinear ways, with different nonlinearities introduced at each biological scale. It is difficult to predict the effect pharmacological perturbations in the transition rates of a multi state ion channel will have on the firing rate of a cell with the channel. A change in the release properties of synaptic vesicles can have multiple non-intuitive effects on the responsiveness of the postsynaptic cell. Clustering of different types of conductances on different regions of a dendritic tree can allow a cell to perform complex nonlinear computations on spatially segregated inputs. These effects highlight the range of electrical, molecular and physical processes through which information is transformed in the nervous system, and the difficulties of inferring from a change in one level of the system the effect on the next. Well structured, data driven computer models encoding our assumptions of the behaviour of the different elements of the system are valuable tools towards gaining an insight into these complex multiscale interactions.

Before going onto more detail on the creation and analysis of multiscale models of neuronal systems and the software infrastructure available to support them, it is useful to try to classify the wide range of approaches used in neuronal modelling.

1.2.1 Classifying different approaches to neuronal modelling

There are many endeavours which can be termed neuronal modelling (Dayan, 2006), each focussing on different biological scales of the nervous system, using diverse assumptions on the core model elements, and various approaches to the analysis of the system behaviour (e.g. connectionist models of cognitive processing, firing rate models of connected brain areas, multicompartmental conductance based models)[1]. To help classify these activities, a number of levels or scales have been devised to distinguish the approaches. These are summarised in Figure 1.1.

[1]And consequently many different strands of research which are called Computational Neuroscience

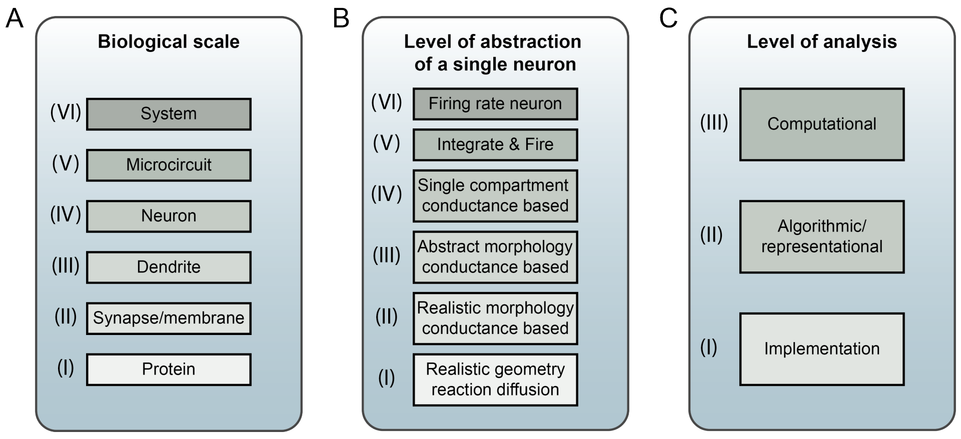

A
Biological scale
(VI) System
(V) Microcircuit
(IV) Neuron
(III) Dendrite
(II) Synapse/membrane
(I) Protein
B
Level of abstraction of a single neuron
(VI) Firing rate neuron
(V) Integrate & Fire
(IV) Single compartment conductance based
(III) Abstract morphology conductance based
(II) Realistic morphology conductance based
(I) Realistic geometry reaction diffusion
C
Level of analysis
(III) Computational
(II) Algorithmic/ representational
(I) Implementation

Figure 1.1: Three possible ways which can be used to categorise neuronal modelling. A) The biological scale of elements included in the model, ranging from interactions at the protein level to systems level behaviour. B) The level of abstraction of the individual neuronal elements used, depending on how much of the physical, electrical and chemical properties of the cell are incorporated into the model. C) The level at which the function of the system is being analysed, whether it is the high level computation that the system is performing, the algorithm that is being used to process information and the way data is represented, or the specific implementation of the system in neuronal "hardware".

A: Biological scales

A key way to classify neuronal models is by the biological scale of the elements included in the model (Figure 1.1A; Shepherd (2004)). Neuronal systems process information at multiple physical scales and can involve gene regulation mechanisms and protein signalling pathways (I), conductance changes at synapses and distributed across membranes (II), computations within dendritic subbranches (III) or across the neuron as a whole (IV).

Network processing can be carried out by local microcircuits (V), consisting of a small number of identified cell types carrying out a specific function, or at a systems level (VI), where whole brain regions may be used as the basic interacting units. Models of neuronal systems usually concentrate on one or a small number of these levels, and make approximations about the behaviour of the system at other levels. Multiscale neuronal models can be defined as models which incorporate biological detail at two or more of these levels.

B: Level of abstraction of individual neurons

Many neuronal models contain representations of individually active cells, and the level of abstraction of these towards or away from a complete biophysical and electrical representation of the cell is another way to categorise such models (Figure 1.1B; Gerstner and Kistler, 2002; Herz et al, 2006). The highest level of biophysical detail (I) involves models incorporating the full 3D structure of the neurons, the electrical field potential inside the membrane, the diffusion and reaction of ions and molecules within this volume, and potentially the explicit location of membrane proteins (Blackwell, 2006; Coggan et al, 2005). Due to the high computational demands of such models, it is more usual that only small sections of cells, e.g. spine heads, are modelled at this level of detail. Conductance based compartmental models are more commonly used to simulate the electrical behaviour of cells in response to ion channels and synaptic input across the cell membrane. These can range in detail from cells based on morphological reconstructions (II), abstract multicompartmental cells (III, with a reduced number of compartments used to investigate electrical behaviour in neurites) to single compartment cell models (IV, where the neuron is assumed to be electrically compact).

Integrate and Fire neurons (V) are a broad class of abstract neuron models which normally have just one or two state variables representing the behaviour of the cells (i.e. membrane potential and potentially an adaptation or recovery variable). Realistic features of neurons including spiking, leaky integration of synaptic input and refractory periods are used to create an approximation of the electrical behaviour of cells, and more complex models have a small number of parameters which when set appropriately lead to voltage responses which mimic various classes of real neurons (Brette and Gerstner, 2005; Izhikevich, 2003)[2]. Cells can be represented as having continuous valued firing rates (VI), and networks constructed where interactions depend on these rates and varying synaptic weights between cells.

The correct choice of level of abstraction of the neuronal elements in a model will be important and will crucially depend on the type and quality of experimental data available related to the phenomena being investigated.

C: Levels of analysis of neuronal models

A third way of distinguishing between models of what the nervous system is doing is based on the work of David Marr (Marr, 1982; Marr and Poggio, 1977). This approach developed from his work on understanding the visual system and describes some complementary levels of analysis for understanding how information is processed in neuronal systems (Figure 1.1C). The highest level (Computational) is that describing the general computation being performed (What is being computed? Why is this being done?). The second level (Algorithmic/representational) addresses how the computation is carried out, seeking a description of the algorithm used and how the required data is represented and manipulated. The final level deals with the specific implementation of the operation in terms of neuronal structures.

[2]This level of abstraction of neuronal model should also include the classic simplifications of the Hodgkin Huxley model such as the FitzHugh-Nagumo reduced model (FitzHugh, 1961) and the Morris-Lecar model (Morris and Lecar, 1981).

These three distinct ways of classifying neuronal modelling approaches will be useful in specifying the types of modelling addressed in this thesis, and will be referred back to later. While a subset of each scale has been focussed on in this work, it goes without saying that mathematical and computational models at all levels are valuable tools towards a greater understanding of brain function.

1.2.2 Why model with high anatomical and biophysical detail?

The tools and modelling languages which are the subject of this thesis facilitate creating and analysing models at level I shown in Figure 1.1C. Models developed at this level are ideally accompanied by a clear description of the computational task being carried out by the system (level III) and the high level algorithm and data transformations being employed (level II). An iterative process of refinement of the description of the system at each of these levels of analysis is a feature of the development process of a good model (Gurney and Humphries, in press). One key question to ask then is what level of detail is most appropriate for the description of the "hardware" implementation of the model. Given that there are multiple types of neuronal models with varying degrees of complexity (Figure 1.1B), why not try to explain the behaviour of the system with the simplest base units possible? If simplified neuron models can be tuned to replicate the firing behaviour of a host of different cell types (Brette and Gerstner, 2005; Izhikevich, 2003) and random networks of Integrate and Fire cells can carry out complex signal processing operations (Gerstner and Kistler, 2002; Vogels and Abbott, 2005) why should we include all of this extra anatomical and biophysical detail along with the multitude of potentially unconstrained parameters it would bring? Why shouldn't the principle of Occam's razor be applied and the simplest conceptual model be preferentially used?

Ion fluxes across membranes have been known to underlie the electrical behaviour of neurons since the pioneering work of Bernstein at the turn of the 20th century (Seyfarth, 2006), and much work has been done to determine the properties of the multitude of active conductance underlying these (Hille, 2001). The distinct set of ion channels present on the membrane can determine the shape of the action potential and the stereotypical response of the cell to synaptic input (Bean, 2007), as well as whether the cell is spontaneously active or not. Both the rate at which a neuron fires for a given input and the precise spike timing can be dependent on the types of ion channels present. The foundation for subsequent model formalisms for describing ion channel behaviour is the Hodgkin Huxley formalism (section 1.2.3; Hodgkin and Huxley, 1952).

Most experimental cellular neuroscientists would agree that treating neurons as point processes has limited ability to capture the range of information processing capabilities of single neurons (Johnston and Narayanan, 2008; Koch and Segev, 2000; London and Häusser, 2005; Silver, 2010). Neurons exhibit a striking array of dendritic morphologies (Figure 1.2). Passive features of dendrites alone lead to a range of possibilities for synaptic integration (London and Häusser, 2005; Rall and Agmon-Snir, 1998) while the active channels in dendrites confer enormous computational power on cells (Johnston and Narayanan, 2008), allowing communication of somatic activity back to remote synapses (Stuart et al, 1997), compartmentalisation of dendritic computation (Poirazi et al, 2003), direction selectivity (e.g. in blowfly visual system (Single and Borst, 1998)) and coincidence detection (e.g. in chick auditory system (Agmon-Snir et al, 1998)).

The wide range of cell types in many brain regions can often be classified according to morphological properties, which in turn have an influence on the potential firing patterns of the cells (Mainen and Sejnowski, 1996; van Ooyen et al, 2002; Vetter et al, 2001). The set of ion channels present on cells and their specific distributions across the dendrites (Figure 1.2) will also be crucial to determining the "neuronal phenotype" (Migliore and Shepherd, 2002). Cable theory was the main conceptual breakthrough for understanding passive current flow in dendritic trees, and forms the basis for the development of compartmental models of neurons incorporating active dendritic conductances (section 1.2.4).

Axons from identified cell types can project to very specific anatomical layers, and hence selectively target dendritic regions of the postsynaptic cells (Klausberger and Somogyi, 2008; Lübke and Feldmeyer, 2007). This type of selective connectivity in three dimensions will clearly have an impact on the processing of synaptic inputs in the postsynaptic cells. Such layered connectivity is a key feature of cortical areas of the brain including the neocortex, hippocampus and cerebellum. More and more experimental data is being obtained on the connectivity patterns of these regions (Douglas and Martin, 2004; Thomson and Lamy, 2007), increasingly coupled with functional measurements of the cells involved (Ko et al, 2011).

Computational models which take into account all of these aspects including active membrane conductances, realistic dendritic morphologies and complex 3D connectivity will be required to fully understand the information processing capabilities of these of brain regions (Segev and London, 2000). An example of detailed modelling closely tied to anatomical and physiological experimental data is found in Vervaeke et al (2010). This investigation into electrically coupled Golgi cells used electron microscopy to

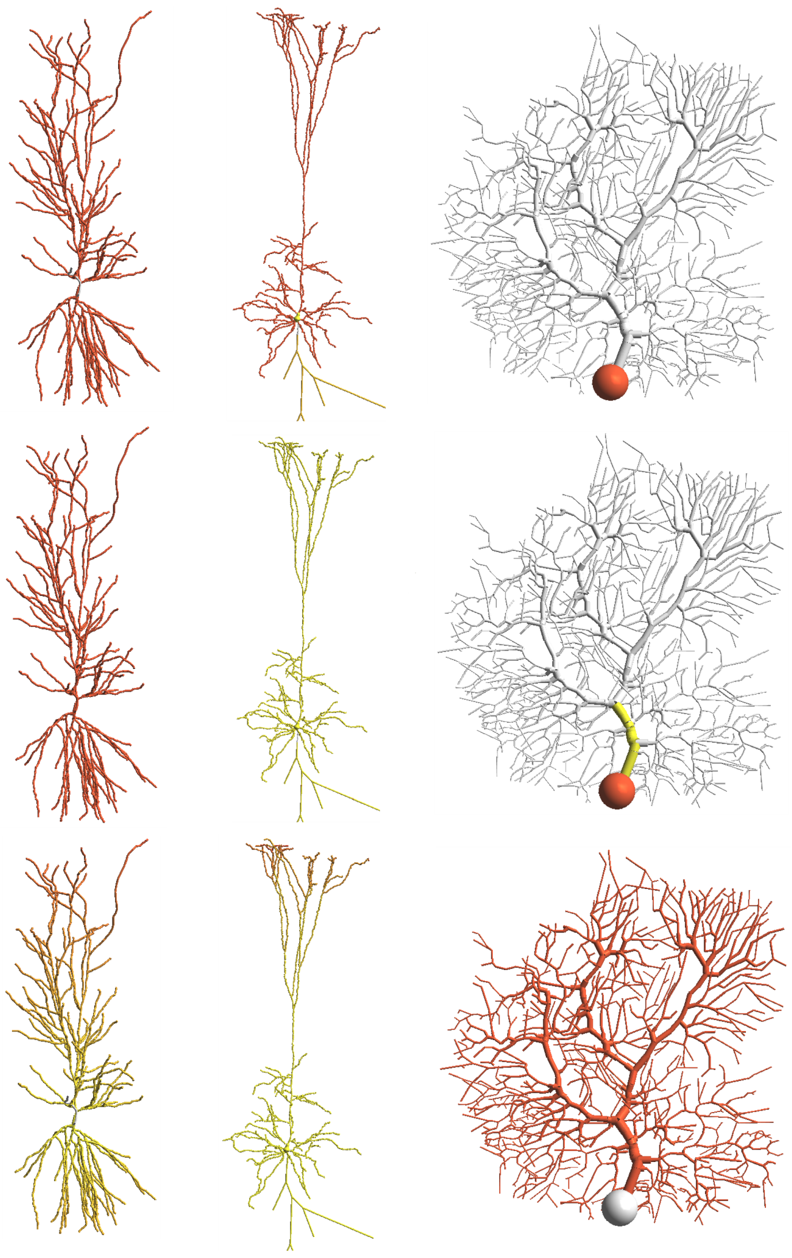


Figure 1.2: Cells from different brain regions exhibit distinct morphologies and possess different complements of active membrane conductance. Left column: CA1 pyramidal cell, with locations of conductances underlying fast Na^+ (top), delayed rectifier K^+ (middle) and I_h (bottom) currents. Red indicates high density, yellow low and grey not present. Middle: Layer 5 pyramidal cell, with (from top) fast Na^+ , Kv1.1 and I_h conductances. Right: Purkinje cell, with (from top) persistent Na^+ , A-type K^+ and P-type Ca^{2+} conductances.

localise gap junctions on the cells' dendrites, paired recordings to characterise the electrical coupling strength, and how it decays with inter-soma distance, and morphological reconstructions for detailed conductance based cell models. These data, together with realistic cell density information, was used to construct a 3D model of the Golgi cell network in a small patch of the cerebellar granule cell layer. This model provided valuable insights into the underlying causes of network desynchronisation by sparse synaptic input which would have been difficult to gain from experiments alone (see section 5.1.4 for more details).

1.2.3 Modelling active conductances

Even today most of the computational models of active membrane conductances used for neuronal modelling rely on the formalism developed in the 1950's by Hodgkin and Huxley. The development of this model for action potential generation in the squid giant axon (Hodgkin and Huxley, 1952) is one of the first and best examples of applying a rigorous mathematical modelling framework to a problem in biology and can be seen as a forerunner of much of computational biology.

Their work led to an understanding of how action potential generation and propagation is based on voltage dependent, ion selective conductances in the axon membrane. Through a series of voltage clamp experiments, they examined this voltage dependence and determined that conductances could be described in terms of a number of "gates", which can be open or closed, all of which need to be open for current to flow. By conducting experiments in normal intracellular fluid, and with zero sodium, they isolated the behaviours of the sodium and potassium components and found that the gates could be activating (increasing probability of being open as the membrane becomes depolarized) or inactivating (more open at hyperpolarized potentials). Gate opening also showed a time dependency for getting to a steady state at a given membrane potential. The functional forms of the expressions for the conductance of a population of sodium (3 activating gates, m and 1 inactivating, h) and potassium (4 activating gates, n) channels (in terms of the total possible conductance, gmax, when all channels are open) are given below:

$$G_{Na}(v,t) = gmax_{Na} \cdot m(v,t)^3 \cdot h(v,t) \tag{1.1}$$

$$G_K(v,t) = gmax_K \cdot n(v,t)^4 \tag{1.2}$$

The gating variables of the Na$^+$ channel, m and h, will be determined by the forward (closed to open, α_{m} and α_{h}) and reverse (open to closed, β_{m} and β_{h}) transition rates (a similar relation applies for n, α_{n} and β_{n}):

$$\frac{dm(v,t)}{dt} = \alpha_m(v) \cdot (1-m) - \beta_m(v) \cdot m \tag{1.3}$$

$$\frac{dh(v,t)}{dt} = \alpha_h(v) \cdot (1-h) - \beta_h(v) \cdot h \tag{1.4}$$

The equations for α_{m}, β_{m} etc. were experimentally determined to have the following forms (using the modern convention of extracellular space at 0mV):

$$\alpha_m(v) = \frac{0.1 \cdot (v+40)}{1 - e^{\frac{v+40}{10}}} \tag{1.5}$$

$$\beta_m(v) = 4 \cdot e^{\frac{v+65}{-18}} \tag{1.6}$$

$$\alpha_h(v) = 0.07 \cdot e^{\frac{v+65}{-20}} \tag{1.7}$$

$$\beta_h(v) = \frac{1}{e^{\frac{v+35}{-10}} + 1} \tag{1.8}$$

An important point about the work of Hodgkin and Huxley is that the model was constructed to match the experimental data they had available at the time, yet the form of the model made a number of suggestions about the biophysical underpinnings of the membrane currents which have since been experimentally verified (e.g. individual membrane proteins selective to ion species containing multiple independent gating elements). This is the hallmark of a good model: it is driven by experimental data and makes predictions which can in turn be experimentally verified. Good models will survive as new experimental data accumulates, confirming their validity, bad models will be dropped. If a relatively simple model such as this can adequately approximate very complex biophysical processes (complex 3D conformational changes of large proteins embedded in the membrane) 60 years after it was first introduced, it is the sign of a very useful model indeed.

There have been many developments in the modelling of ion channels since this pioneering work, not least in terms of the computational options available for solving the equations for the model (Brette et al, 2007). While Hodgkin and Huxley had to use a hand cranked calculator to produce solutions to the equations, a range of neuronal simulators are available today to researchers to simulate neural networks containing hundreds of thousands of such cells (section 4.1.1). There have been significant developments too in terms of the biophysical description of the ion channels themselves (Destexhe and Huguenard, 2000; Hille, 2001). An important extension of the Hodgkin-Huxley model was incorporation of intracellular [Ca^{2+}] dependence on gating behaviour, as found in BK and SK channels. It is also more common nowadays for ion channel physiologists to describe the multitude of states a channel can be in and the set of transition probabilities to change between these (known as kinetic state or Markov models). These transitions can be complex functions of membrane potential and temperature, and depend on [Ca^{2+}] or a number of other channel specific ligands and computational models of channels are increasingly including these features (Destexhe and Huguenard, 2000; Graham, 2002).

Despite these advances the basic Hodgkin-Huxley formalism is still widely used in conductance based neuronal modelling. The question of how much further detail is really needed will very much depend on the question being addressed with the model (Fink and Noble, 2009). Investigation of the effect of channelopathies or a specific drug interaction on the behaviour of a network will require a greater level of detail at the individual ion channel level, but will lead to greater computational costs for the simulations. Standardised ways to share the information on ion channel kinetics between researchers who wish to use them in computational models have been lacking until recently.

1.2.4 Cable theory and compartmental modelling

The elegant dendritic processes that most neurons possess were sadly neglected for many years by experimentalists as merely passive summators of synaptic input, with all of the interesting computations happening at the soma. This idea slowly began to change as researchers gained more insight into the electrical properties of dendrites and the unique computational capabilities they possessed. Inspired by work investigating the propagation of electrical signals through undersea telegraph cables, researchers in the 1950s and 1960s, in particular Wilfred Rall (Rall, 1959; Rall and Agmon-Snir, 1998) started to develop models of current flow in dendrites which took into account the axial flow through the conducting cytoplasm, the ion flow

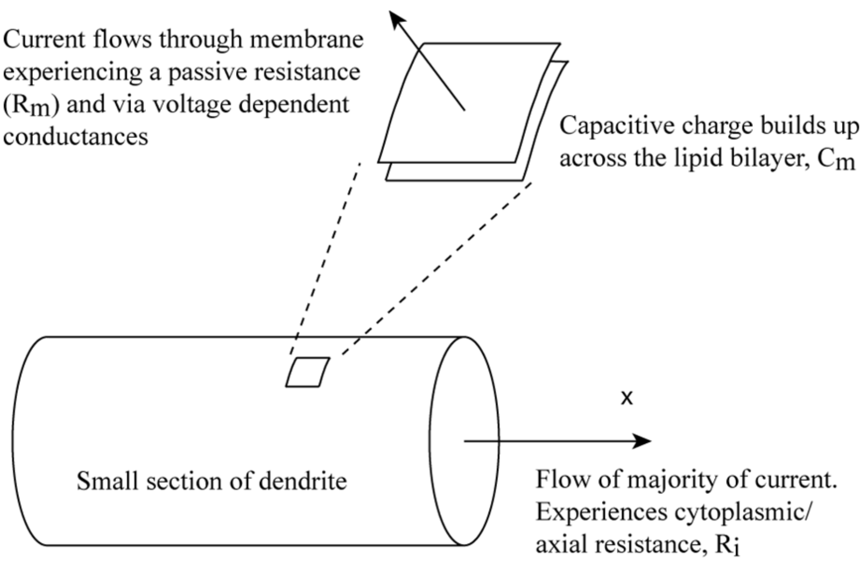

Current flows through membrane experiencing a passive resistance (Rm) and via voltage dependent conductances
Capacitive charge builds up across the lipid bilayer, Cm
x
Small section of dendrite
Flow of majority of current. Experiences cytoplasmic/ axial resistance, Ri

Figure 1.3: A section of a dendrite illustrating the flow of electrical current

through the membrane and the capacitive charging of the lipid bilayer. By considering these currents in small sections of the dendrites (Figure 1.3), it was possible to derive expressions for the flow of electrical current and hence the spread of membrane depolarisation in a a one dimensional dendrite in terms of these experimentally measurable quantities.

The second order partial differential equation below describes the time varying membrane potential, v, at a given point x along an infinite dendrite.

$$\frac{1}{R_i}\frac{\partial^2 v(x,t)}{\partial x^2} = C_m\frac{\partial v}{\partial t} + \frac{v}{R_m} \tag{1.9}$$

Defining the terms $\lambda = (\mathrm{R_m}/\mathrm{R_i})^{1/2}$ (space constant) and $\tau_\mathrm{m} = \mathrm{R_m C_m}$ (membrane time constant) allows the equation to be rewritten as (Rall, 1959):

$$\lambda^2\frac{\partial^2 v(x,t)}{\partial x^2} - \tau_m\frac{\partial v}{\partial t} - v = 0 \tag{1.10}$$

The concepts of the space constant (and hence the electrotonic length, physical length divided by λ) and the membrane time constant are still very relevant today. The pioneering work in this area led to many predictions for the properties of passive dendrites, including the greater attenuation of signals travelling towards the soma as opposed to towards the distal dendrites, the broadening and delaying of synaptic potentials, and the basic rules of integration of distributed excitatory and inhibitory synaptic input.

While this basic formalism is sufficient to explain a number of features of passive dendrites and to provide analytical solutions to the time evolution of currents in them, it is insufficient in the case where the dendrites are endowed with active voltage dependent conductances. In this case, compartmental modelling is used (Segev and Burke, 1998), where the cell broken into isopotential compartments (Figure 1.4) and the synaptic and membrane currents into and out of each of them is calculated. This reduces the problem to a set of ordinary differential equations which needs to be solved for the behaviour of the system. This has required specialised software packages and efficient algorithms (Hines, 1984) to be developed. Today, there are a number of options for developing detailed compartmental models (section 1.3.1) to investigate the complex interplay of morphology and active conductances underlying dendritic computation. However, each uses its own approach to solve the systems of equations and until recently it has been very difficult to test that model behaviour is independent of the choice of numerical integration method used.

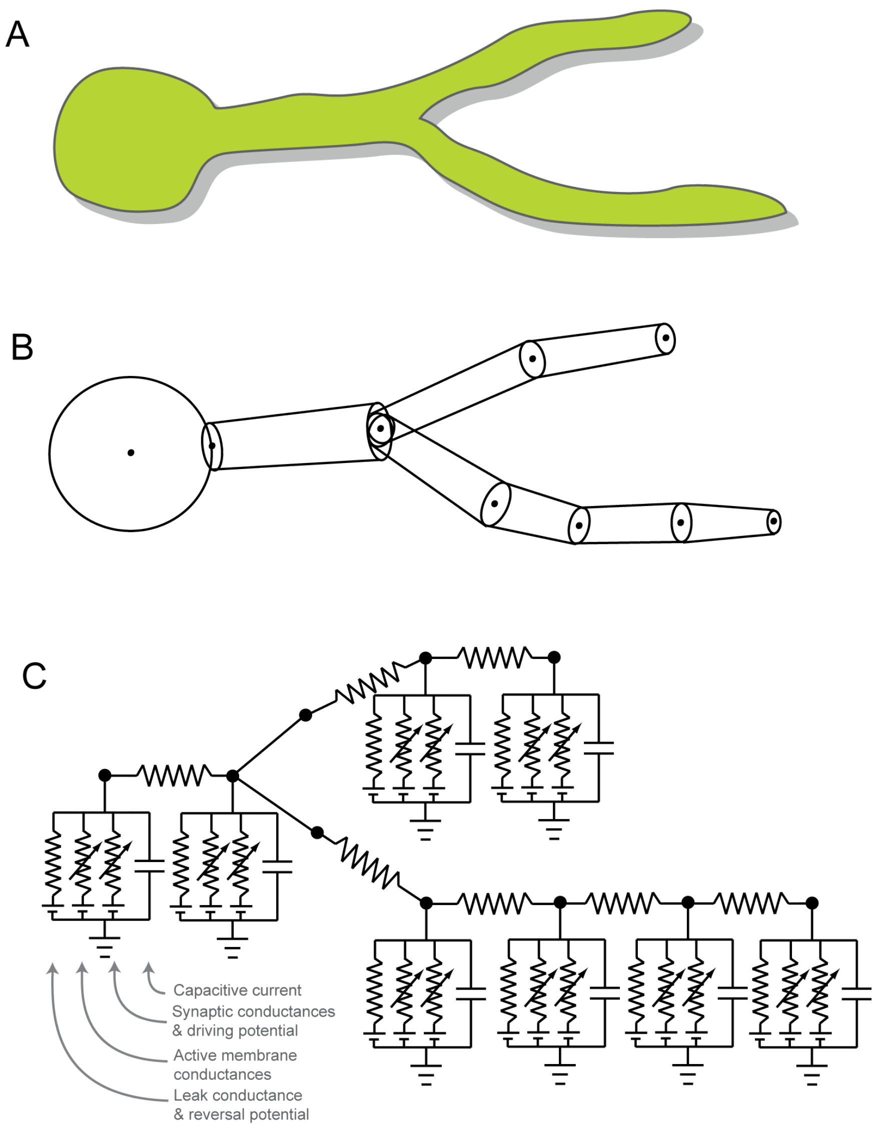


Figure 1.4: Modelling of dendrites as electrically connected compartments. A) Schematic of original neuron whose electrical behaviour is to be modelled. B) Compartmentalisation of neuronal morphology. The cell is split into a small cylinders with dimensions based on the original cell. C) Equivalent circuit representation of the cell. Each compartment is modelled as a sub-circuit with a capacitance, ohmic leak current, and time varying currents due to active conductances across the membrane and synaptic conductances. The compartments are linked by resistances due to the cytoplasmic resistivity.

1.3 Current practices in detailed neuronal modelling

Some of the key model formalisms used when creating detailed neuronal models have been discussed in the previous section. But how are these used in practice? What computational resources have been made available to the community to facilitate building and analysing such models? How much computational experience is needed to create and modify models with such packages? What happens to a model after a publication is released describing research conducted using it? Does it get reused or developed further by other parties? And how have these technical and social aspects of model development contributed to the lukewarm reception that biophysically detailed modelling often receives from experimental neuroscientists?

1.3.1 Neuronal simulators

Creating a program to simulate a network of Integrate and Fire neurons requires a relatively small amount of code (a few tens of lines of MATLAB should suffice[3]) and most researchers using such networks normally develop their own system from scratch, customised to their needs and preferred model structure. In contrast, simulators for conductance based neurons (beyond the classic Hodgkin-Huxley squid axon model), potentially capable of multicompartmental modelling, take longer to create and test, and development of these normally follows one of three routes: 1) The core numerical integration platform (potentially with graphics and analysis functionality) evolves into a general purpose neuronal simulation platform, of use to others outside the initial developers. Examples given below generally fall into

[3]See example here: http://www.izhikevich.org/publications/spikes.htm

this category, with strong user communities, online documentation, and lists of publications available where the simulators are used; 2) The simulation platform is used mainly by the core set of modellers who initially created it for their own research and while it is made freely available is little used outside of that team or beyond the models it was initially developed for (e.g. Roger Traub's FORTRAN implementation for his thalamocortical network model (Traub et al, 2005), Lyle Graham's SurfHippo[4]); 3) The core simulation platform continues to be developed and used in ongoing research but is not made widely accessible to the community (e.g. Paul Rhodes' simulation platform (Rhodes and Gray, 1994), Izhikevich thalamocortical network model (Izhikevich and Edelman, 2008), SPLIT simulator (Djurfeldt et al, 2008), Blue Brain Project infrastructure (Markram, 2006)). While this last scenario may sometimes be due to the desire to keep a competitive advantage over others in the field, it can also arise due to lack of time/manpower to support any other users of the system.

Thankfully, an increasing number of software packages are being made freely available to the wider community. The advantages of having multiple users testing and helping to refine a package can help make up for the extra work needed to support the software. An important part of this process is making the applications open source. Users are free to inspect the source code of the application to debug, optimise or extend the package, all of which benefits the wider user base.

Two of the most widely used packages for detailed neuronal modelling have been NEURON (Carnevale and Hines, 2006) and GENESIS (Bower and Beeman, 1997). Both of these packages support development of conductance based multicompartmental cell models and networks of neurons (of types II-

[4]http://www.neurophys.biomedicale.univ-paris5.fr/∼graham/surf-hippo.html

IV in Figure 1.1B, and sometimes type V), and have been in development for over 20 years. NEURON is still in active development with important recent extensions including the ability to run on parallel computing resources (Hines and Carnevale, 2008; Migliore et al, 2006). Active development of GENESIS 2 has stopped but there are important reimplementations of the core features in MOOSE (Multiscale Object Oriented Simulation Environment; Ray and Bhalla (2008)) and Neurospaces/GENESIS [5] (Cornelis and De Schutter, 2003). One other recently developed simulator for multicompartmental modelling is PSICS (Parallel Stochastic Ion Channel Simulator, Cannon et al (2010)) which allows simulation of detailed neuronal models which include stochastic ion channel transitions, and so can be used to examine the effect of low numbers of ion channels on neuronal firing behaviour.

Another widely used simulation platform is NEST (NEural Simulation Tool, Diesmann and Gewaltig (2002)). This platform is mainly intended for large scale networks of simplified neurons (type V in Figure 1.1B) and the continued development of it has led to much valuable research in the general principles of network simulations on parallel hardware (Morrison et al, 2005; Plesser et al, 2007). Brian (Goodman and Brette, 2008) is a Python based simulator which specialises in allowing model behaviour to be specified in simple scripts, and is actively developing methods to use these scripts to generate optimised code for faster simulation, including execution on graphics processing units (GPUs) and other parallel computing hardware (Goodman, 2010). XPP (or XPPAUT, these names are used interchangeably) is a widely used tool for the analysis and simulation of generic dynamical systems, and can be used to construct a wide range of single compartment cell models (of types IV-VI in Figure 1.1B), well as small networks of these (Ermentrout,

[5] http://www.genesis-sim.org

2002).

This brief overview illustrates that there are multiple packages available for the simulation of neurons at different levels of abstraction (Brette et al, 2007). The relative strengths of each of these platforms in different technical areas will determine the choice of which to use for a specific modelling task and these have been discussed in more detail elsewhere (Gleeson et al, 2010b). While the availability of a range of simulators has had many advantages, there have also been problems associated with this diversity (Djurfeldt and Lansner, 2007). A key issue has been that the format for the native scripts for each of the simulators are different, e.g. a cell morphology file or a channel model created for NEURON cannot be used directly by GENESIS, making it difficult to reuse models and check their behaviour across simulators.

1.3.2 Model publication, dissemination and reproducibility

A key requirement for any scientific publication is that it should contain enough information for another researcher in the field to independently reproduce the described results. In cases where obtaining experimental data depends on new or specialised equipment this can be difficult in practice, but for publications focussed on modelling, giving access to the simulation scripts used is an excellent way to allow others to reproduce, critically assess and build on the results obtained in the paper.

In practice however, this is not always how things proceed. Journals (even ones focussed on computational neuroscience) generally do not require release of simulation scripts when modelling results are published. This is in contrast to bioinformatics publications, which require for example submission of microarray data in a recognised database before publication[6]. There is still a requirement for modelling papers to give sufficient detail to reimplement the model (e.g. large tables of ion channel kinetics, channel densities, network connectivity matrices, etc.) but in practice it is quite difficult to fully reproduce the model simply from data given in the paper and supplementary information. While there has been some work on producing guidelines for information in publications on network models (Nordlie et al, 2009), there is still no substitute for freely available scripts which can be rerun to reproduce figures from a publication.

ModelDB[7] (Hines et al, 2004) has been a valuable resource for those looking for scripts from publications using computational models. It contains a large number of neuronal models built using a range of simulators (with the NEURON simulation environment being the most popular) and it has been instrumental in convincing the community of the usefulness of providing code to accompany published models. While there are still some researchers who don't release their modelling code after publication, the benefits in terms of others referencing and building on one's work are incentive enough to convince most researchers to share their simulation code.

1.3.3 Accessibility and model reuse issues

Assuming that the scripts for a model described in a paper are available, there are still some barriers to wider reuse of a published model (or components of it, for example channels, synapses or individual cell models). With most simulators having their own proprietary scripting formats, a researcher must either use that simulator or manually convert the model to their pre-

[6]A list is presented here: http://www.mged.org/Workgroups/MIAME/journals.html of journals requiring MIAME (Minimum Information About a Microarray Experiment) compliant data as a condition for publishing papers detailing microarray based results

[7]http://senselab.med.yale.edu/modeldb

ferred platform's format (and so become proficient in both simulators' native languages). Success at that endeavour usually depends on well structured and commented code, something which is not always the case. If the simulator is "home grown" or not widely used, even basic documentation for the simulator scripting language itself might be lacking.

Modularity of the code is needed for extracting specific components of a model, but is not always present, e.g. in MATLAB scripts where the physiological properties of a model might be interspersed with numerical integration setting, simulation control flow, visualisation elements and analysis functions[8].

These issues point to the need to develop a modular exchange format for describing the physiological components of detailed neuronal models. This should not contain any information specific to particular simulator's implementation of the model, but just the parameters to create an instance of a model according to a shared model formalism. Also, the exchange format should not need to restate the underlying model equations each time (e.g. each channel file shouldn't need to redefine the Hodgkin-Huxley model). The specification of the format should describe the physiological concepts sufficiently and each application should map this to/from its internal format when importing/exporting the model. In this way a researcher can choose one tool for constructing the models, choose the fastest or the one with best parallel computing support for simulating the model, and the most appropriate for visualising and analysing the results.

Reuse of models in this way has other advantages too. Testing a model on multiple simulators can remove any dependencies of the model behaviour

[8]An example of this practice can be found in the following model: http://senselab.med.yale.edu/ModelDb/ShowModel.asp?model=59479; a cerebellar stellate cell model in MATLAB specifying cellular properties, channel kinetics, numerical integration and plotting of results, all in a single file but with just 3 lines of comments

on the implementation of the original simulator. This is very useful for debugging the models which are quite complex software systems in themselves. The more researchers who use the models means more critical eyes on the model parameters and behaviour. Researchers who wish to use model components which have been tested by multiple groups can have more confidence to use them as "black boxes" on which to build larger scale composite models.

1.3.4 Biological variability versus mean model development

A common practice when creating a cell model has been to tune the model to a particular set of experimental data, e.g. the average measured physiological values (such as passive electrical properties) or spiking behaviour. This does not take account of the variability inherent in biological systems (Marder and Taylor, 2011) where a wide range of cellular behaviours can be exhibited by the cells of the same type. Similarly for network models, identical network elements are normally used and often there is no heterogeneity in the cellular or synaptic parameters.

An alternative approach to producing a single ideal model is to create populations of models, and investigate how well these models reproduce the measured ranges of behaviour of the experimental system. It has been found that models with disparate sets of parameters can reproduce target behaviour (both in the case of the conductance densities in single cell models (Golowasch et al, 2002) and when varying synaptic properties in networks (Prinz et al, 2004)). This suggests that in biological systems too there is no one ideal set of channels or synapse properties which are needed to produce a desired system behaviour.

Most of the models shared on ModelDB have fixed sets of model parameters representing “mean” model behaviour and reproduce a single aspect of the published model (e.g. one of the figures from the paper). While these “snapshots” of models are useful entities for sharing and encoding in standardised declarative formats, it is also useful to share the code for analysing them, for generating parameter searches and multiple instances of network connectivity. This is ideally done with a multi-platform, easy to learn, modern programming language, as opposed to scripts tied to the original simulation platform. Facilitating investigation of a model’s behaviour across its biological parameter space can maximise the knowledge shared when such models are made available to the community.

1.3.5 Generation and control of large scale network simulations

A particular set of issues arises when exchanging models whose complexity requires high performance computing resources to build, simulate and analyse them. While a morphologically detailed single cell model or small network can be easily run on a single processor and a graphical interface provided to plot membrane potential and other variables, managing the construction and execution of large scale networks in a parallel computing environment, and dealing with and extracting useful information from the simulation results, is usually very specific to the hardware and software infrastructure involved. Those researchers who develop the system, set up the compute nodes and manage the simulations are often the only ones who can get full benefit from the system as a whole.

In some cases groups of researchers who develop such systems do make concious efforts to share their work with others, as in the case of the NEST

Initiative[9], who make their core simulator available and have many publications on the general principles of managing large scale simulations (Morrison et al, 2005; Plesser et al, 2007). In other cases the complexity of the specialist hardware infrastructure involved and the complex tool chain needed for the iterative development of large scale models has been used as a reason for not actively sharing software e.g. in the case of the Blue Brain Project (Markram, 2006).

In any case, there are still significant barriers to the use and analysis of such complex networks for researchers without backgrounds in software development. This is a particular concern since it is precisely investigators who have hands on experience with the physiology and anatomy of the systems in question who will be the ones who will assess how "realistic" these network models are and who will be needed to contribute their knowledge to improving the systems.

1.3.6 Testing and validation of distributed model components

When distributing software models to other researchers, (automated) testing of model components will also be important. How does someone who downloads a particular model know that it's behaving as intended on their operating system, with their particular version of the simulator? If they port the model to another simulator what are the key results that need to be reproduced? While these types of tests are standard practice in software development, they are rarely used with computational neuroscience models, but inclusion of these would increase the confidence of researchers who want to reuse elements of existing models.

[9]http://www.nest-initiative.org

1.3.7 Comparison with Systems Biology field

The Systems Biology[10] field (Kitano, 2002) has been developing and sharing models of signalling pathways, metabolic and gene regulatory networks in a more organised fashion as compared to computational neuroscience for many years (De Schutter, 2008). A number of years ago the developers of some of the tools for modelling and analysing such networks came together and decided to come up with a "lingua franca" to exchange the models between their applications, and SBML (Systems Biology Markup Language) was created (Hucka et al, 2003). Now over a decade old, it is in its third major revision, there are over 200 software tools which support all or part of the language[11] and there is a growing database (BioModels[12]) containing curated models in the format (Le Novère et al, 2006), each of which can be easily validated as complying to a specified version of the language.

An initiative with similar aims, CellML (Lloyd et al, 2004), is being developed by the Auckland Bioinformatics Institute and also has a number of supporting applications[13] and a database of models (Lloyd et al, 2008) with entries covering a wide range of biological phenomena. While CellML is developed by a smaller core team than SBML, it has close links with the Virtual Physiological Human project (Kohl and Noble, 2009). CellML and SBML have developed as separate initiatives but there have been efforts at technical interoperability for some time (Schilstra et al, 2006). CellML versions of most of the SBML models in the BioModels database can be down-

[10]There are almost as many definitions of Systems Biology as there are of Computational Neuroscience. Here I use the definition of the field which emphasises the construction and validation of computational models as a key element in a workflow of iterative refinement of models of a biological system: initial experimental data is gathered on a system, conceptual and computational models are created which lead to new predictions for the behaviour of the system, experimental verification is performed and models refined

[11]http://sbml.org/SBML_Software_Guide

[12]http://www.biomodels.net, with 366 curated models (September 2011)

[13]http://www.cellml.org/tools

loaded. There are also efforts at having joint meetings (and "hackathons" aimed specifically at software interoperability testing) between these and other initiatives in the field[14].

A number of valuable lessons can be learned from experiences in the Systems Biology field which would increase the accessibility, interoperability and reuse of models in computational neuroscience. However, the multiple biological scales in neuronal systems (Figure 1.1), the spatially distributed nature of neuronal signalling and the host of nonlinear interactions between entities in the nervous system substantially increase the challenges for creating standards for and sharing neuronal and network models.

1.4 Outline of work

To address the issues raised in the previous sections, I have been actively involved in the development of new tools to facilitate the development of complex networks of biophysically detailed neuronal models. The work was initially motivated by the need to create more detailed 3D cerebellar network models within the Silver Lab (Gleeson et al, 2007; Vervaeke et al, 2010), but the generally applicability of the tools and methods have led to wider interaction with the computational neuroscience community and close involvement with international efforts to improve the overall interoperability and usability of software for modelling neuronal systems. My contributions in these areas are summarised below.

[14]Coordinated by the COmputational Modeling in BIology NEtwork: http://www.co.mbine.org

Software for greater accessibility of neuronal models

To facilitate the development of networks of biophysically detailed neurons, I developed the application neuroConstruct (Gleeson et al, 2007). It features a graphical user interface (GUI) to allow complex 3D networks to be constructed and visualised, supports code generation for many widely used simulators, and has inbuilt network analysis functionality. Chapter 2 discusses the motivation, design considerations and functionality of neuroConstruct in more detail.

This open source application is intended for use both by researchers without extensive programming experience for automated model generation and analysis, and for more advanced modellers who want much lower level control over model creation and simulator execution. A number of published cell and network models have been converted to neuroConstruct format and made available for users of the application. The software was publicly released in 2007 and to date has had over 1,100 registered user downloads.

Contribution to standards development for biophysically realistic neuronal modelling

The multitude of simulator specific formats for creating neuronal models has been a concern of the community for many years (Brette et al, 2007; Cannon et al, 2007; Djurfeldt and Lansner, 2007). NeuroML[15] has been an initiative dedicated to developing specifications for describing models in computational neuroscience for over a decade. I became involved in this process early in my PhD and have been one of the main technical contributors to the initiative since then. Version 1.x of the language focussed on describing networks of multicompartmental conductance based neuron mod-

[15] http://www.neuroml.org

els (Gleeson et al, 2010a), and was heavily influenced by efforts to generate matching simulation results on NEURON and GENESIS from code generated by neuroConstruct. Over time NeuroML models could be run on other simulators in this way, and a number of other software developers have independently added NeuroML support (Gleeson et al, in press). Currently there are over 20 software packages which support some part of NeuroML and others have support in their development roadmaps[16].

Efforts towards NeuroML version 2.0 are also well under way. This new version will have greater flexibility in model specification for extending the language, have better support for abstract models (types V-VI in Figure 1.1B) and will allow greater interaction with languages such as SBML and CellML (allowing models to be created across scales I-VI in Figure 1.1A).

Chapter 3 describes the development of the NeuroML language, discusses version 1.x in detail, outlines the tools with NeuroML support I have developed, and describes ongoing work for version 2.0 of the language.

Creation and management of large scale network simulations

The types of models which can be created with neuroConstruct and exchanged in NeuroML format span from single compartment cell models to complex 3D microcircuits containing thousands of cells. The large scale networks enabled by these technologies cannot be run on single processors and require interaction with high performance computing platforms. neuroConstruct was extended to allow generation of scripts for Parallel NEURON, enabling simulation of networks on a much larger scale than previously possible. This has been incorporated so that the interaction with high performance computing resources is as transparent as possible: there is minimal

[16]An up to date list of all packages featuring NeuroML support is maintained here: http://www.neuroml.org/tool_support

extra interaction through the GUI needed for simulations to run on remote distributed hardware as compared to running them locally.

Another valuable extension of the core neuroConstruct functionality was the addition of a scripting interface based on the Python language. This language is becoming widely used in computational neuroscience as a method of tying specialised software packages together. Providing this interface to neuroConstruct facilitates complex repetitive operations and generation of populations of models which would normally have to be done through the GUI.

Chapter 4 discusses the various additions to neuroConstruct and associated tools which I have created to facilitate management of large scale simulations.

Making key cell and network models from diverse brain regions available in standardised and accessible formats

A number of detailed cell and network models have been converted to NeuroML format over the past number of years. These include detailed cell models from the cerebellum (De Schutter and Bower, 1994; Solinas et al, 2007a; Vervaeke et al, 2010), network models of the granule cell layer (Maex and Schutter, 1998), morphologically detailed cortical (Kole et al, 2008; Mainen et al, 1995) and hippocampal (Migliore et al, 2005) pyramidal cell models, a network model of the dentate gyrus (Santhakumar et al, 2005) and a thalamocortical network model (Traub et al, 2005) containing neuron models from multiple cortical layers and the thalamus. The process of converting these models has highlighted a number of issues with models developed for specific simulators and has helped to optimise the supporting simulators. The models themselves have been more thoroughly tested and often extended beyond their original scope (e.g. network models converted to 3D). Having models in a common format has also allowed direct comparison of models of the same cell types developed by different groups on different simulators.

Chapter 5 details the range of models which have been studied as part of this work, a number of which have been used for original scientific research within the Silver Lab (Farinella et al, 2011; Rothman et al, 2009; Vervaeke et al, 2010). These have been made available as core examples of models in NeuroML and as neuroConstruct projects and have developed and improved as these technologies move forward.

Chapter 6 summarised the main objectives of this work and evaluates the performance of the solutions presented against these. Chapter 7 discusses the work in the wider context of computational neuroscience and the ongoing efforts to understand the brain through a combination of experimental, theoretical and computational investigations.

Chapter 2

Design and Implementation of neuroConstruct

This chapter describes neuroConstruct (Gleeson et al, 2007, 2008), a modular software application for the development of detailed neuronal and network models in 3D. It explains the motivation for developing the application, gives details on the implementation, describes the key features, and outlines the interaction of the application with the range of supported simulators and other tools.

2.1 Overview

2.1.1 Motivation for developing neuroConstruct

The main motivating factors for developing neuroConstruct have been both biological and modelling related, i.e. what are the anatomical and physiological aspects which need to be incorporated into the models, and what methods have been available for creating them?

Biological perspective

A wide range of information processing tasks can be carried out in complex dendritic trees endowed with active conductances. Electrical behaviour can be very different in distinct regions of the cells, and reconstructed neuronal morphologies coupled with information on ion channel placement and kinetics allow investigation of this. Experimentalists also need to be able to relate the behaviour of these models with the data they have recorded in real cells.

Many interesting regions of the brain have multiple cells of different types arranged in distinct 3D patterns. More experimental data is becoming available on the simultaneous behaviour of populations of cells in such areas. Models of signal flow and information processing in these networks will be of greater use for understanding neuronal function if significant details of their 3D structure can be incorporated into the models.

Neurons in many cortical regions exhibit complex connectivity patterns, with layer and cell region specific synaptic connectivity. Much of this is nonuniform, with probabilities of connection and synaptic strengths decaying with distance. Allowing models to be created which incorporate such complex cells and connectivity, and which can be visually verified and analysed (particularly important for models with large numbers of cells and connections) was the initial motivation for the application.

Modelling perspective

Creation of detailed cell and network models has typically involved writing scripts in a simulator specific format. These scripting languages commonly allow a mixture of model creation, execution logic, graphical elements and simulation analysis. This approach is useful for a single researcher or closely working group of modellers, but can result in quite a complex software system when the model becomes large. It also makes it difficult for other parties to reuse all or part of the model.

Even for an experienced modeller, visually verifying network structure and complex connectivity patterns in a network model incorporating 3D cellular elements and anatomical based connectivity patterns was difficult until recently. Both NEURON (Carnevale and Hines, 2006) and GENESIS (Bower and Beeman, 1997) have graphical user interfaces allowing visualisation of cells and networks, but for the most part these have remained little changed since first being added to the tools about 15 years ago.

Development of neuroConstruct

neuroConstruct has been developed around an advanced 3D graphical interface for interacting with complex neuronal models in a modular fashion (Figure 2.1). The cerebellar cortex with its regular structure of stereotypically repeated cells was the initial target system, but the application has been used for modelling a wide range of other brain regions. While NEURON was the first simulator to be supported, it soon became clear that the bulk of the cell structures, channel representations and network information stored in neuroConstruct for this simulator could be mapped also to GENESIS. This mapping was added and allowed direct comparison of simulation behaviour between independently developed simulators for the first time. This has become one of the main drivers for the development of neuroConstruct, and provided a very useful platform for the development of the current structure of NeuroML (Gleeson et al (2010a), chapter 3).

Key driving factors behind the ongoing development

The following are the main driving factors which guide the ongoing development of neuroConstruct:

- **3D visualisation:** Allowing visual inspection of 3D cell structure and network connectivity is important for comparing models to the measured anatomy.

- **Modularity:** Model elements such as cells, channels and synapses should be easily transferable between projects to facilitate sharing and reuse.
- **Accessibility:** Human readable descriptions of model elements are key to increasing the accessibility of the physiological components of the models as compared to the formats used in existing simulators.
- **Interoperability:** Importing from and exporting to multiple different formats facilitates use of neuroConstruct in a wide range of researchers' toolchains.
- **Cross simulator validation of models:** Complex neuronal models can easily acquire behaviour which is dependent on a particular simulator's implementation so being able to test the same model on multiple independently developed platforms is crucial for quality control.
- **Extensibility:** As existing simulators and visualisation/analysis tools get updated or new applications get developed, the core neuroConstruct platform should be extensible to allow interaction with these new features.

2.1.2 Implementation aspects

The core of neuroConstruct is implemented in the programming language Java[1]. The main advantage of this language is that there is a Java Runtime Environment available on multiple operating systems (including Windows, Linux and Mac). Classes for building graphical user interfaces are included

[1]http://java.sun.com

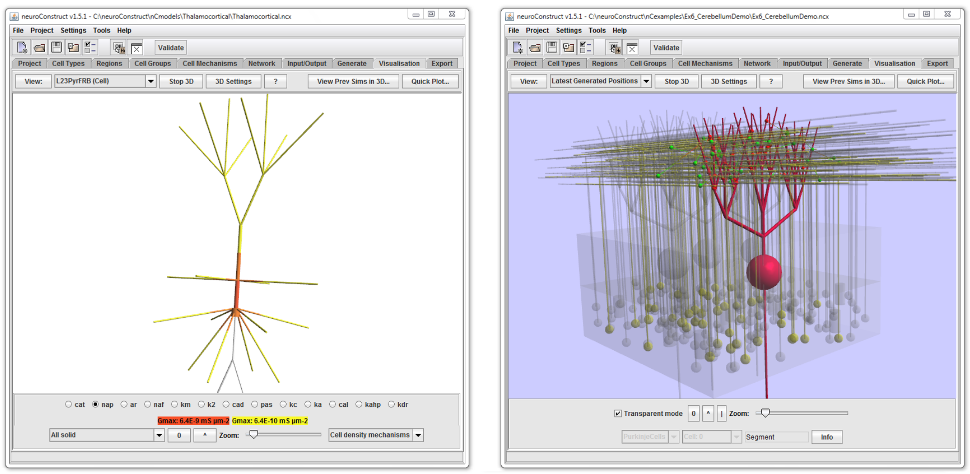

File Project Settings Tools Help
Validate
Project Cell Types Regions Cell Groups Cell Mechanisms Network Input/Output Generate Visualisation Export
View: L23PyrFRB (Cell)
Stop 3D 3D Settings ? View Prev Sims in 3D... Quick Plot...
cat nap ar naf km k2 cad pas kc ka cal kahp kdr
Gmax: 6.4E-9 mS µm-2 Gmax: 6.4E-10 mS µm-2
All solid
Zoom:
Cell density mechanisms
View: Latest Generated Positions
Transparent mode
Segment
Info

Figure 2.1: Two screenshots of the main neuroConstruct graphical user interface. Image on left shows a single cell with reduced number of compartments (Layer 2/3 Pyramidal cell from Traub et al (2005)) with an Na^+ channel conductance density that varies on different parts of the cell membrane. Image on right shows the visualisation of a simplified cerebellar network model using the transparency feature to highlight a single cell and its connections.

with the core of the language, facilitating a common look and feel across operating systems.

A number of freely available libraries are available for Java, and those used for neuroConstruct include Java3D[2] for visualisation of network structure, JUnit[3] for unit testing (section 4.5), Java HDF5 tools[4] (section 2.3.2) and Jython[5] for the Python based scripting interface (section 4.4). All of the Java bytecode and native libraries for these libraries are included with the standard release, and while this makes for a larger download (~40 MB) it facilitates execution across the above mentioned platforms. The application is open source and consists of approximately 130,000 lines of Java code, the

[2]https://java3d.dev.java.net
[3]http://www.junit.org
[4]http://hdf.ncsa.uiuc.edu
[5]http://www.jython.org

vast majority of which I have written myself. Details on downloading and installing neuroConstruct are outlined in section 2.4.1.

2.1.3 Project structure

Models that are created in neuroConstruct are organised into projects. A project contains details on the cells, cell groups[6], channels and synapse mechanisms, network connections and electrical inputs which can be used in simulations. All files for the project (including simulation data) are contained in a directory named after the project.

A project is normally intended to illustrate a number of aspects of the behaviour of its cells and networks and multiple **simulation configurations** are defined, each with a certain combination of cell groups, network connections, inputs etc. together with information on what data to plot and/or record during the simulation. These simulation configurations can be generated independently and can be used to illustrate each cell's individual behaviour, normal or pathological network state, etc. or can reproduce each of the figures in an accompanying publication.

2.2 Key features of neuroConstruct

The functionality of neuroConstruct can be grouped into five main areas (Figure 2.2).

- **Import and validation of neuronal morphologies:** Reconstructed neuronal morphologies, commonly used in conductance based neuronal models (Ascoli, 2006), can be imported into neuroConstruct in various formats (e.g. Neurolucida format) and automatically checked for er-

[6]The term population can also be used for cell group

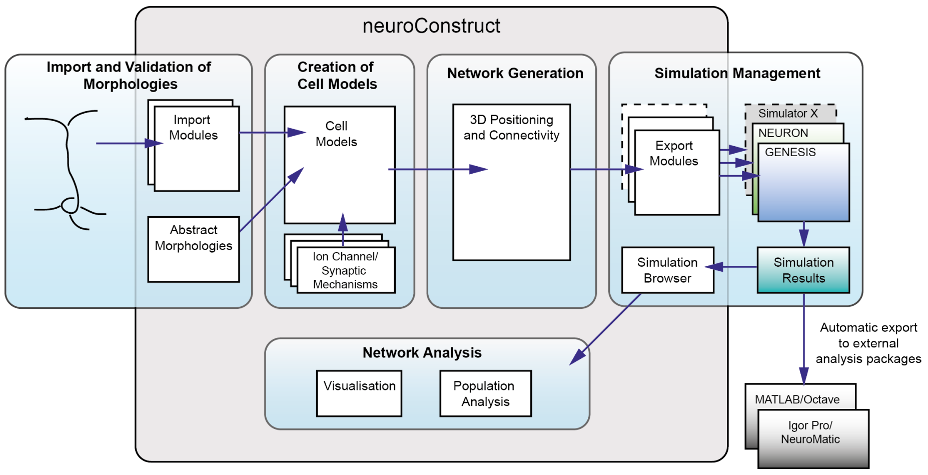


Figure 2.2: An overview of the core neuroConstruct functionality. Reproduced from Gleeson et al (2007).

rors. Reduced neuron models with a smaller number of compartments (or just single compartments) can also be created manually (e.g. Figure 2.1, left) by specifying the 3D locations and diameters of the cell sections. Cells can be visualised with various levels of detail depending on the graphical capabilities of the host machine (section 2.2.3).

- **Creation of simulator independent conductance based cell models:** Modelling of active membrane conductance changes produced by voltage and ligand gated ion channels is essential for reproducing the complex spiking behaviour of real neurons (Segev and London, 2000). Channel mechanisms can be defined in neuroConstruct in a simulator independent format (ChannelML, part of the NeuroML language) and cell models created by specifying the complement and density of these conductances on the cell surface (Figure 2.1, left).

- **Automated network generation and visualisation:** Once cell models have been created in neuroConstruct, they can be placed within a region of 3D space at a specified density (Figure 2.1, right). Layered structures can be created from stacks of contiguous regions to recreate the anatomical layout of regions such as the neocortex or cerebellum (Shepherd, 2004). Once the cells are arranged in 3D, synaptic connections can be stochastically generated according to specified sets of rules.

- **Simulation Management:** Network simulations are carried out by automatically generating script files for a number of simulator packages such as NEURON and GENESIS with the results (membrane potential traces, channel conductances, etc.) stored in text or binary files. Simulations in these packages run independently with no interaction with neuroConstruct.

- **Network Analysis:** Simulations can be loaded back into neuroConstruct for visualization and analysis. For more specialized analyses, script files are created that facilitate the data being imported into two common numerical analysis packages (MATLAB and Igor Pro).

These features allow creation, visualisation, simulation and analysis of detailed cell and network models, placing minimal requirements on a user's programming experience. Lowering the barriers to usage of such detailed models was key novel aspect of this work. Each of these functionality areas are discussed in more detail in the following sections.

2.2.1 Neuronal morphologies in neuroConstruct

Neuronal models with complex morphologies have been used to investigate various aspects of synaptic integration and neuronal excitability (De Schutter and Bower, 1994; Destexhe and Pare, 1999; Jarsky et al, 2005; Mainen et al, 1995; Migliore et al, 1995; Poirazi et al, 2003; Vetter et al, 2001), and public databases have been produced that contain examples of anatomical reconstructions of stained neurons (Ascoli et al, 2007; Cannon et al, 1998). However, using such morphology files in compartmental models is complicated by the fact that they are often in different formats, their anatomical and electrical compartments are not equivalent and there are subtle differences in how the morphological information is used by different simulators. To overcome these problems neuroConstruct can import and visualize morphology files with different formats, including Neurolucida[7], GENESIS ***readcell*** compatible format[8], most NEURON/ntscable generated morphology files, Cvapp (SWC) format (Cannon et al, 1998) and MorphML (Crook et al, 2007).

There are a number of similarities between each of these ways of representing neuronal morphologies (which are all primarily structured lists of 3D points with diameters) and the main differences are due to the focus of the original application which developed the format (e.g. visualisation/reconstruction of anatomical features vs. compartmental modelling). An important part of the development of neuroConstruct was creating a simulator independent representation of neuronal morphology based on a superset of these, which allows translation between simulator specific formats.

A schematic example of a simple cell (Figure 2.3A) digitized in Neurolucida format is shown in Figure 2.3B. An outline is used to represent the shape of the soma, but the remainder of the neuron is described using three-dimensional points with associated diameters along branches. Nodes are points at which new branches start. This information is encoded in a

[7]http://www.mbfbioscience.com/neurolucida
[8]http://www.genesis-sim.org/GENESIS/Hyperdoc/Manual-25.html#ss25.131

hierarchical tree in Neurolucida ASC files. Much more information can be present in a Neurolucida file (e.g. paths in space to distinguish anatomical layers), due to the fact that the format is designed to encode any interesting anatomical features found in the optical images being reconstructed, though most of this information is not relevant for most applications interested in single cell reconstructions.

In NEURON, unbranched neurites are specified by sections consisting of sequences of 3D points and diameters (sections are between blue points in Figure 2.3C). Membrane surface area and axial resistance are computed from this information. While full anatomical detail is used for surface area, numerical integration is only carried out at points determined by the parameter ***nseg*** (number of segments). If ***nseg*** is 1, the section is considered isopotential. If ***nseg*** is greater than one the sections are split into that number of segments (normally much lower than the number of 3D anatomical points) and membrane potential etc. is calculated at each of these points (red points in Figure 2.3C).

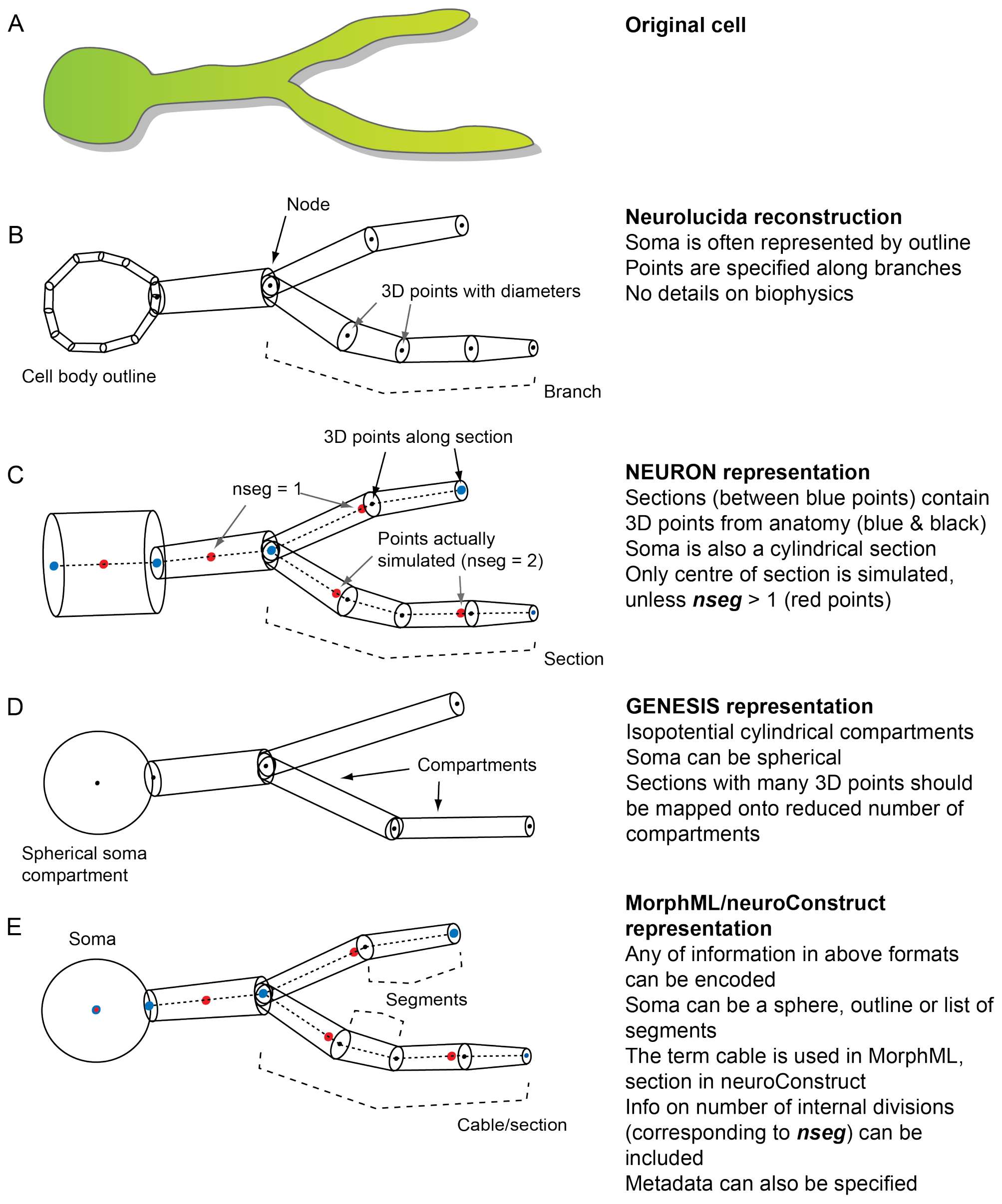
GENESIS uses a single isopotential unit, termed a compartment, as the building block for both morphology and numerical integration (Figure 2.3D). Since the number of 3D anatomical points is much greater than the number of compartments required for adequate spatial discretisation (normally determined by a maximal allowable electrotonic length of any compartment), some recompartmentalisation of the cell structure should be carried out.

The internal representation of the morphology used in neuroConstruct is closely related to MorphML (Crook et al, 2007), a language for describing neuronal morphologies. MorphML is based on XML (eXtensible Markup Language), and is the core of Level 1 of the NeuroML framework (Crook et al (2007); Gleeson et al (2010a); chapter 3). neuroConstruct defines a section (which maps directly onto a NEURON section) as an unbranched part of the neuronal morphology (between blue points in Figure 2.3E) and these contain one or more segments, whose endpoints (black points) give the 3D structure along the section. For historical reasons the term section is used for this in neuroConstruct, while cable is used in MorphML (a segment in neuroConstruct is also a segment in MorphML). The ***nseg***/spatial discretisation value (red points) is a property of the neuroConstruct section.

The close relationship between the cell descriptions in neuroConstruct and MorphML (and also for higher Levels of cell description, e.g. including biophysics) allows most cells to be exported from neuroConstruct in pure NeuroML and reimported with no loss of information.

Mapping of a cell in neuroConstruct onto NEURON format is straightforward. Sections are mapped to NEURON sections and 3D point information is provided, while the number of spatial discretisation points is used for ***nseg***. With GENESIS, each segment is mapped to a compartment by default. However, this can lead to too many points being calculated along dendrites however and can slow simulations.

To address this issue, I implemented a new feature to recompartmentalise neurons, allowing morphologies originally based on detailed neuronal reconstructions (e.g. ∼1000-10000 segments) to be mapped onto a reduced number of segments/GENESIS compartments (∼100-1000). As illustrated in Figure 2.4, overall cell structure is preserved, and each section (e.g. lower dendrite in B with 18 segments) is mapped onto two single segment cylindrical sections which can be used as GENESIS compartments. The radii of the cylinders are calculated to preserve cell membrane surface area, total length and axial resistance along sections. The number of compartments to generate for long sections is determined by a maximum desired electro-



Original cell

Neurolucida reconstruction
Soma is often represented by outline
Points are specified along branches
No details on biophysics

NEURON representation
Sections (between blue points) contain 3D points from anatomy (blue & black)
Soma is also a cylindrical section
Only centre of section is simulated, unless ***nseg*** > 1 (red points)

GENESIS representation
Isopotential cylindrical compartments
Soma can be spherical
Sections with many 3D points should be mapped onto reduced number of compartments

MorphML/neuroConstruct representation
Any of information in above formats can be encoded
Soma can be a sphere, outline or list of segments
The term cable is used in MorphML, section in neuroConstruct
Info on number of internal divisions (corresponding to ***nseg***) can be included
Metadata can also be specified

Figure 2.3: Various formats for representing neuronal morphologies. A) Illustration of cell to be reconstructed. B) Typical Neurolucida reconstruction of cell. C) NEURON representation of cell using sections. D) GENESIS representation of cell with reduced number of compartments. E) neuroConstruct and MorphML representations of the cell. Modified from Crook et al (2007).

tonic length[9]. This feature has been important in getting simulations of the same model in NEURON, GENESIS and MOOSE to execute in comparable times. There are some important issues to consider however with the balance between speed of simulation and spatial discretisation of complex cells (section 5.11).

Automatic checks in neuroConstruct signal potential problems with morphologies including dendritic segments of zero diameter or zero length and dendrites that are detached from the cell body. Manual editing of the imported morphologies is possible e.g. to fix highlighted problems. Large scale networks of thousands of neurons often use simplified cell models with fewer compartments to minimize the computational overhead (Santhakumar et al, 2005; Traub et al, 2005), and such abstract cell models can be created by adding sections and segments to a simple cell model.

While other applications have been developed to convert neuronal reconstructions between different morphological formats[10], neuroConstruct is unique in being so closely integrated with the simulation platforms for modelling the neurons.

2.2.2 Creation of conductance based cell models

Both morphologically detailed cells and abstract (or single compartment) cells can be used as a basis for spiking cell models, by specifying the passive electrical properties (specific membrane capacitance and leak conductance, axial resistance) and by providing the membranes with voltage and/or intracellular ligand gated conductances and external synaptic input.

[9]More details of how to access this functionality through the GUI is available here: http://www.neuroconstruct.org/docs/Glossary_gen.html#Compartmentalisations

[10]For example NLMorphologyConverter at http://neuronland.org.

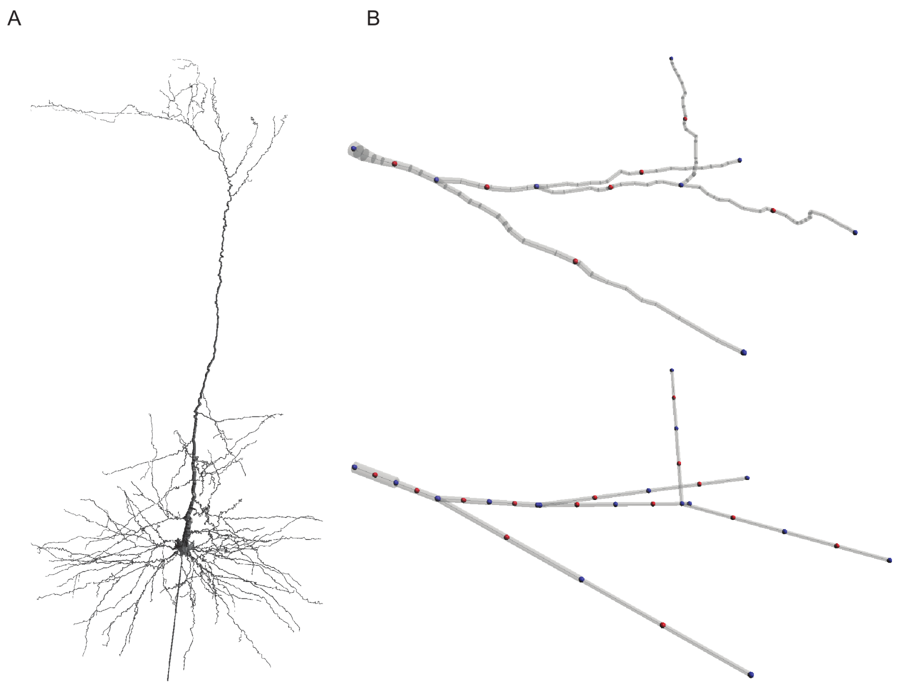


Figure 2.4: Detailed cell morphologies in neuroConstruct. A) A detailed reconstruction of a neocortical pyramidal cell (Mainen et al, 1995) imported into neuroConstruct from a NEURON morphology file. B) Detail of a small part of a dendritic tree. Upper view: all 3D detail present in the original morphology file. Sections (between the blue spheres) contain a number of 3D points with associated diameter, each of which is the endpoint of a segment (small grey conical frusta). NEURON uses this information to compute membrane area and axial resistance, but only performs numerical integration at specific locations (red spheres; determined by ***nseg***). Lower view: recompartmentalised representation of cell structure with fewer segments for mapping to GENESIS. Reproduced from Gleeson et al (2007).

Simulator independent descriptions of ion channel mechanisms

Neuronal signalling is mediated by a variety of subcellular, membrane and synaptic mechanisms. Models of cellular mechanisms can be simple, such as synaptic conductance waveforms, or more complex like Hodgkin-Huxley type formulations of voltage-gated conductances (Hodgkin and Huxley (1952), section 1.2.3), which depend on both voltage and time, with conductance densities which can be nonuniformly distributed over the cell membrane (Migliore and Shepherd, 2002). Such models form a core part of any conductance based neuronal simulation, but their implementation is one of the more complicated aspects of using existing simulation packages. Although the mathematical framework used to describe such mechanisms (e.g. maximum channel conductance, reversal potential, rate equations) is general and familiar to many neuroscientists, implementation of these in simulators such as NEURON or GENESIS usually involves use of a platform specific programming language.

Models of ion channels and synaptic mechanisms are implemented in neuroConstruct using a ChannelML based description, which forms part of Level 2 of the NeuroML framework. Full details on ChannelML are provided in section 3.3.2 and a brief overview is given here. Figure 2.5 shows a ChannelML file describing a synaptic conductance mechanism, and how it can be used. It consists of an XML file containing the physiological parameters in a structured format that can be validated against a specification (in an XML Schema Definition file, section 3.3.3), reducing the probability of errors of omission.

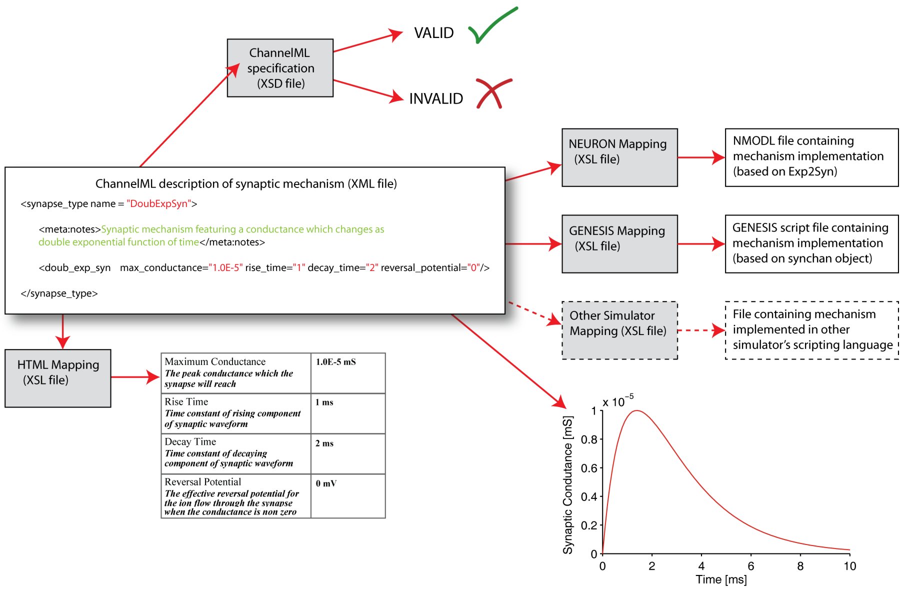
Information in XML files can easily be transformed into other formats with an XSL (eXtensible Stylesheet Language) mapping file. I created XSL files which map ChannelML descriptions of cell mechanisms onto NMODL

(Hines and Carnevale, 2000) format for NEURON and onto the appropriate channel/synapse object in a GENESIS script file. The simulator-independent XML format promotes future compatibility with other simulators: for each newly supported simulator, a single XSL file needs to be created which maps the files onto its specialized format (as has been the case with ion channels for PSICS (Cannon et al, 2010)). The nature of XML also allows translation of the file into HTML, allowing the cell mechanism to be presented in an easy to read format, facilitating online archiving.

A number of ChannelML ion channel and synapse templates are included with neuroConstruct. These examples can be used as starting points for new channel and synapse models, and the XML file can be edited directly or the parameters can be modified through the GUI. In this way they can be updated to match the channel kinetics in a particular cell type, either from a published model or directly from experimental measurements of these parameters (e.g time course and steady state of m and h for a Hodgkin-Huxley type model of an Na^+ channel). To allow for unsupported models and to provide greater backwards compatibility, files in NMODL (*.mod files) or GENESIS script (e.g. based on the ***tabchannel*** object) can be incorporated into cell models created with neuroConstruct, although this makes the model simulator specific. This can facilitate the translation of a mechanism from a simulator specific format to ChannelML[11].

Differences between systems of units used in a model descriptions in papers and those supported by simulators are a frequent source of errors when reimplementing detailed neuronal models. GENESIS uses a consistent set of either SI units or physiological units (based on ms, mV and cm), whereas NEURON has its own system based on physiological units (Table

[11]Step by step details of the process of converting a channel model to ChannelML are given here: http://www.neuroconstruct.org/docs/importneuron.html


ChannelML specification (XSD file)
VALID
INVALID
ChannelML description of synaptic mechanism (XML file)
NEURON Mapping (XSL file)
NMODL file containing mechanism implementation (based on Exp2Syn)
GENESIS Mapping (XSL file)
GENESIS script file containing mechanism implementation (based on synchan object)
Other Simulator Mapping (XSL file)
File containing mechanism implemented in other simulator's scripting language
HTML Mapping (XSL file)
Synaptic Condutance [mS]
x 10^-5
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
Time [ms]

```
<synapse_type name = "DoubExpSyn">

    <meta:notes>Synaptic mechanism featuring a conductance which changes as
    double exponential function of time</meta:notes>

    <doub_exp_syn   max_conductance="1.0E-5" rise_time="1" decay_time="2" reversal_potential="0"/>

</synapse_type>
```

Maximum Conductance ***The peak conductance which the synapse will reach***	**1.0E-5 mS**
Rise Time ***Time constant of rising component of synaptic waveform***	**1 ms**
Decay Time ***Time constant of decaying component of synaptic waveform***	**2 ms**
Reversal Potential ***The effective reversal potential for the ion flow through the synapse when the conductance is non zero***	**0 mV**

Figure 2.5: An example of a NeuroML file and the methods available to validate it and transform it into other formats. This example shows the description of a synapse model and the parameters needed to specify its double exponential time course synapse. An XSD (XML Stylesheet Document) file is used to check the file is a valid part of NeuroML (in this case ChannelML). The file can be converted into script files in the native language of various neuronal simulators, using an XSL (eXtensible Stylesheet Language) file for each mapping. HTML representations of the XML file provides a more readable view of the mechanism and associated metadata. Plots can be generated to view the mechanisms properties. The correct form of the ChannelML file (what attribute names to use in the elements, etc.) is specified in an XSD (XML Schema Definition) file. See section 3.3 for more details on the structure of NeuroML. Modified from Gleeson et al (2007).

2.1). Conversion between these systems of units is handled automatically by neuroConstruct.

	neuroConstruct	**NEURON**	**GENESIS (PHY)**	**GENESIS (SI)**
Time	t **[ms]**	t **[ms]**	time **[ms]**	time **[s]**
Voltage	v **[mV]**	v **[mV]**	Vm **[mV]**	Vm **[V]**
Length	r,x,y,z **[µm]**	diam, L **[µm]**	dia,len **[cm]**	dia,len **[m]**
Current (point process)	i **[nA]**	i **[nA]**	inject **[µA]**	inject **[A]**
Specific capacitance	SpecCap **[µF/µm^2]**	cm **[µF/cm^2]**	CM **[µF/cm^2]**	CM **[F/m^2]**
Resistance (e.g. membrane resistance)	SpecRes x (Area) **[Kohm]**	(1/g) / (PI x diam x L) **[ohm]**	Rm **[Kohm]**	Rm **[ohm]**
Conductance density (e.g. in **pas**)	gmax **[mS/µm^2]**	g **[S/cm^2]**	g **[mS/cm^2]**	g **[S/m^2]**
Specific membrane resistance	SpecRes **[Kohm µm^2]**	1/g **[ohm cm^2]**	RM **[Kohm cm^2]**	RM **[ohm m^2]**
Conductance (point process, e.g. synapse)	gmax **[mS]**	g **[µS]**	gmax **[mS]**	gmax **[S]**
Specific Axial Resistance	SpecAxRes **[Kohm µm]**	Ra **[ohm cm]**	RA **[Kohm cm]**	RA **[ohm m]**
Axial Resistance	SpecAxRes x (Len/Area) **[Kohm]**	Ra x (4 x L/(PI x diam^2)) **[ohm]**	Ra **[Kohm]**	Ra **[ohm]**
Concentration	Conc **[mol/µm^3]**	Conc (mM or milli moles per liter) **[mmol/dm^3]**	Conc **[mol/cm^3]**	Conc **[mol/m^3]**

Table 2.1: Systems of units used in neuroConstruct, NEURON and GENESIS. Note that neuroConstruct's units are neither wholly physiological nor SI, but are designed to be consistent, while using millivolt and millisecond for voltage and time (as in physiological units) and using micrometer as the unit of length (most important quantity for 3D applications). Conductance for point processes in NEURON is out of step with the rest of its units.

Linking channel mechanisms and morphologies in spiking cell models

Once a cellular morphology has been imported or created in neuroConstruct, groups of sections can be defined, for example to classify them as axons, somata and dendrites. Subgroups of these, such as proximal, oblique and apical dendrites, can also be defined. The distribution of cellular mechanisms can then be specified by associating a channel/conductance density pair with a specific section group. For example, a channel can have a non-uniform density across the cell by varying the conductance density in each group (Figure 2.1A). Conductances can also have densities which vary as function of path distance along the dendrites. Ion concentration mechanisms (e.g. activity-dependent intracellular Ca^{2+} pools) can also be added in this way, as can passive electrical properties (specific membrane capacitance and specific axial resistance), allowing spine densities to be simulated without additional compartments. New cell models can be created from experimental or published data using neuroConstruct by importing/creating cell morphologies and modifying existing ChannelML templates or adding native code.

The process of tuning a conductance based cell model to replicate a set of experimental data is a complex one. Manual editing of the model parameters (e.g. conductance densities) through the GUI is possible, but tuning of cell models is better implemented by use of the Python scripting interface to neuroConstruct (section 4.4) to allow complex cell model optimisation algorithms (Druckmann et al, 2007) to be performed.

The simulator independent representation of cell models in neuroConstruct was a major advance in allowing direct comparison of spiking behaviour of equivalent cell models between the main conductance based neuronal simulators, helping improve model quality and benchmark simulator performance.

2.2.3 Network generation and visualisation

Cell models can be placed in specific regions in 3D space and synaptic connectivity generated based on a number of different algorithms.

Cell Placement in 3D

The gross anatomy of a brain region is generated in neuroConstruct by defining 3D regions in which specific cell types are placed. Regions can currently be rectangular boxes, spheres, cylinders or cones and multiple regions can be used to create composite structures such as the layers found in the cerebellum and cortex. Cells in neuroConstruct are arranged in cell groups, which are created by specifying the cell type, the 3D region in which the cells are found and the packing pattern used to fill the space. Possible packing patterns include: cubic close packing for maximum density in 3D space; evenly spaced packing in 3D with cell body centres aligned; hexagonal planar patterns; single cells placed at a specified location; cells placed in a one dimensional line. However, for many brain regions random cell placement is more realistic. The number of cells in a specified region can be set and whether cells should avoid the space occupied by existing cell bodies or can overlap can be specified. This allows cell densities to be matched to experimentally measured values for a particular brain region, e.g. as used in Vervaeke et al (2010) for the 3D Golgi cell network (section 5.1.4), or in the large scale network simulations of the cerebellar granule cell layer (section 4.4).

Synaptic Connectivity Patterns

Once cells are arranged in 3D, synaptic connections can be created between cell groups, or within a single cell group. The set of rules specifying synaptic connections between cell groups and the associated type of synaptic mechanism is termed a network connection. There are two different ways in which network connections can be generated in neuroConstruct. The first, Morphology Based Connections, works by defining regions on pre and postsynaptic cells where synaptic connections are allowed. Several other parameters can also be set including the number of synapses per cell and the maximum and minimum connection lengths. Figure 2.6A shows how the Morphology Based Connection algorithm can be used to generate connections between simplified models of granule cells and Purkinje cells in the cerebellar cortex. Granule cell axons consist of an ascending axon and a T-shape bifurcation giving rise to parallel fibres, which passes through the planar dendritic arbour of Purkinje cells (Figure 2.6Ai-ii). The parallel fibre sections were specified as potential locations of presynaptic connections and a subset of the Purkinje cell dendritic sections (Figure 2.6Ai-ii, red) were specified as possible postsynaptic connection locations. In this case the number of connections between pre- and postsynaptic neurons was constrained to a maximum of one (Figure 2.6Aiii). This type of network connection was used in Rothman et al (2009) to target excitatory and inhibitory synaptic input on to the basal dendrites of a detailed layer 5 pyramidal cell (Figure 5 in that paper, reproduced as Figure 5.10 in this thesis)

The second algorithm, which is termed a Volume Based Connection, is designed for cases where the axon is a dense, highly arborized structure, distributed over a specific region of space, as commonly found in cortical structures (Klausberger and Somogyi, 2008; Lübke and Feldmeyer, 2007; Thomson and Lamy, 2007). Figure 2.6Bi shows a simplified model of a cortical interneuron and a cylindrical volume that defines the bounds of its axonal arborization. The diagram of a simplified pyramidal cell in Figure 2.6Bii shows the subset of its dendritic tree where connections of that type are permitted. When the cells are placed in 3D, regions of the dendritic trees of a number of pyramidal cells which fall within the axonal arbour of the interneurons are potential candidates for connections. These are generated

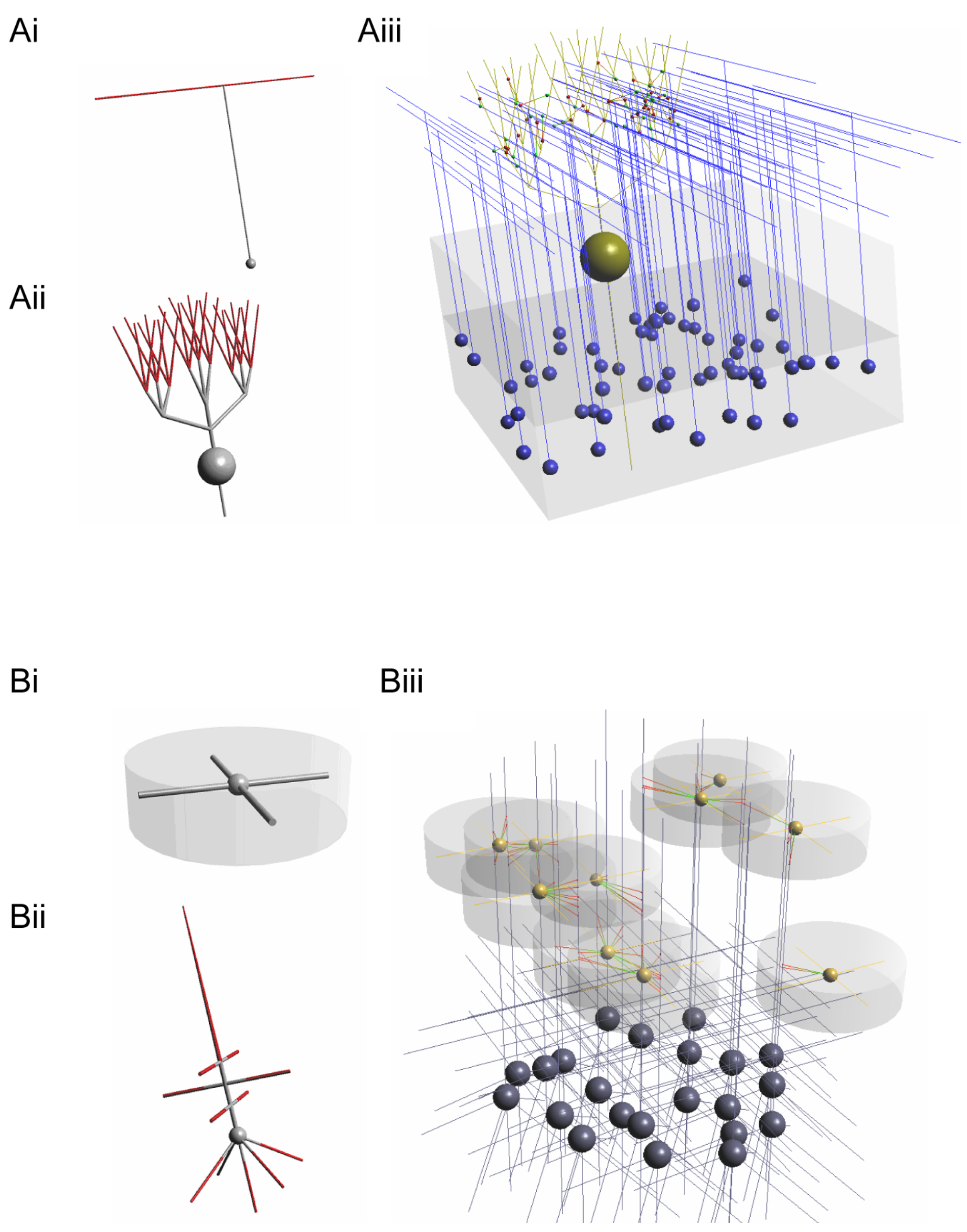


Figure 2.6: Network connection types in neuroConstruct. A) Morphology Based Connections: pre- (i) and postsynaptic (ii) cells showing potential locations for that connection type in red, and a network created using the cells (iii). B) Volume Based Connection: presynaptic cell (i) with surrounding axonal arborisation region, postsynaptic cell (ii), and generated network (iii). Sites of pre- and postsynaptic connection are linked with lines changing from green to red. Reproduced from Gleeson et al (2007).

randomly based on the user-defined connectivity conditions, which include the number of connections per source cell and the maximum number of connections on each target cell. Other shapes including cones and spheres can be used to define the 3D bounds of axonal arborizations. The probability of making a synaptic connection within this volume can also be non uniform, e.g. with a Gaussian profile, allowing a preference for local but a chance of longer range connections, leading to the possibility of a small world network topology (Watts and Strogatz, 1998). There is also provision to introduce randomness into the amplitudes of the synaptic conductances and their onset delays for both connectivity algorithms.

The automated generation of network connections in 3D was one of the key original features of neuroConstruct, and was very difficult to achieve with existing scripting based solutions up to that point. One limitation though, is that the cells are not "grown" to connect presynaptic cells with postsynaptic locations, and that all of the cells in a cell group are identical. There is active research into the growth processes of neurons and a number of computational modelling approaches are used to investigate neural development (van Ooyen, 2011). Two applications which allow generation of neuronal networks based on rules for branching probability and chemotaxis are NETMORPH (Koene et al, 2009) and CX3D (Zubler and Douglas, 2009). The cells and network structures generated by each of these applications can now be exported to NeuroML (section 3.5) and imported into neuroConstruct. This allows researchers to integrate the functionality of these specialist applications into a simulation toolchain with neuroConstruct.

Synaptic properties

Each network connection specifies the type of synapse model to use. These are normally based on ChannelML, but native scripts for simulators (e.g. a NEURON mod file) can be specified. ChannelML allows a range of synapse types to be expressed (section 3.3.2) including AMPA (α-amino-3-hydroxy-5-methyl-4-isoxazolepropionic acid) and GABA$_A$ (γ-aminobutyric acid, class A) receptor mediated synapses and nonlinear voltage-dependent synapses (e.g. NMDA (N-Methyl-D-aspartate) receptor mediated). Synaptic plasticity can be incorporated with short-term plasticity (STP) and spike timing dependent plasticity (STDP) synaptic mechanisms. It is also possible to specify creation of multiple synaptic mechanisms at each connection (e.g. colocated AMPA and NMDA). Electrical synapses at gap junctions can also be created.

A scalar weight factor is generated for each of the synaptic connections, and this can have a fixed or random value, or can be a generic function of distance between connection points or somata. This was an important new feature which enabled spatially heterogeneous connectivity and was used to incorporate experimentally measured data on distance dependence of electrical coupling strengths between Golgi cell pairs into a 3D network model (Vervaeke et al (2010), section 5.1.4).

Stimulation of networks

The external activation of a network with defined patterns of stimuli can be achieved in several ways. Cell groups can receive two main types of inputs: current steps of specific duration, delay and amplitude, or random trains of synaptic inputs, with a defined input frequency or a range of frequencies. Both of these types of input can be applied to all cells in a group, to a fixed percentage of cells, or to cells inside or outside a defined 3D region. The last option can be used to apply spatially organized input patterns to networks.

Extensions of these basic input types allow a generic expression in time to be defined for current amplitude or stimulation rate (e.g. sine wave or ramp), which allow more complex input patterns to be applied to the network.

Generation and visualisation of networks

Simulation configurations are sets of cell groups, network connections and inputs which illustrate different aspects of the cells and network in a neuroConstruct project. Each of these will be of a particular duration and can specify what data to plot and or save during the simulation. When a simulation configuration is specified for generation, a seed is used to generate the random numbers for the cell locations, weight distributions, etc. This seed is recorded and can be used to regenerate the same network at a later date.

After the network is generated, it can be viewed in 3D. As network models can vary widely in size, there are a number of functions in neuroConstruct to facilitate the clear display of large networks, cells with complex morphologies and individual synaptic connections. These include showing the dendrite and axon as lines (Figure 2.6Biii), or just somata with neurites hidden, rather than the full 3D structure of each cell (Figure 2.1A). An adjustable transparency mode is available for visualizing cells deep within large networks. This allows an individual cell, defined groups of cells or cells within a defined region to be highlighted (Figure 2.1B). These functions allow individual cells to be viewed within networks of thousands of cells on most standard desktop computers.

The cell placement and network connectivity can be imported and exported in NetworkML format (Level 3 of the NeuroML framework, section 3.3.2), allowing generated networks to be saved for later display or networks created with other applications to be loaded into neuroConstruct for visualization and use in simulations.

2.2.4 Simulation management

The range of simulators and other export formats to which the generated network can be converted are listed under the tab Export. Presently these comprise NEURON, GENESIS, PSICS, PyNN and NeuroML (more details on each of these options in given in section 2.3.1). These allows generation of native simulator scripts which can then be viewed with an internal text file viewer, or set running. The processes execute independently of neuroConstruct and multiple simulations can be set running at one time. When the simulations finish, each simulator saves the results in a common format (a text file with lists of membrane potentials etc., or a compact binary format based on HDF5 (section 4.3.2)).

The Simulation Browser GUI lists all completed simulations and these can be listed showing simulator, simulation configuration, time of execution and other relevant information.

2.2.5 Network activity analysis

Simulations loaded back in to neuroConstruct can be analysed in a number of ways. Clicking on individual cells selects them and plots of the membrane potential at that point can be created. Depending on what was specified to be saved during the simulation, other parameters (channel conductance, rate variables, etc.) at the soma or any segment of the cell can also be plotted. The menu in the window of these plots offers a number of analysis options including listing spike times, analysing firing rates and generating phase plane plots (dv/dt vs. v).

Populations of cells can also be analysed, and rasterplots and histograms of cell firing can be generated, as well as plots of interspike interval distributions. Plots can also be generated of a selected cell's synchrony with respect to the rest of the cells in the population (cross-correllograms, e.g. Figure 5.5). These features allow rapid analysis of network behaviour without the need to write scripts.

The network activity can also be replayed provided at least one variable is saved from each cell in the network. This can be useful for getting a feel for how the activity is spreading within the network (e.g. between cortical layers). The membrane potential in each segment in a detailed cell can also be saved and replayed, to investigate for example the propagation of an action potential back into a dendritic tree (see Figure 5.9). This option generates a large amount of data for detailed cells, but has the advantage that visualisations of spread of depolarisation can be paused, rewound, etc. in the neuroConstruct GUI.

2.2.6 Advanced features

Some of the advanced features of neuroConstruct which have been added recently include:

- **Parallel code generation:** Support for generating code for Parallel NEURON (Hines and Carnevale, 2008; Migliore et al, 2006) allows much larger simulations to be executed in parallel computing environments (section 4.2).
- **Python interface:** While most parameters for cell and network models can easily be edited through the GUI, the Python based scripting

interface gives maximum flexibility for generating multiple instances of networks or performing parameter searches on models (section 4.4).

- **Greater NeuroML integration:** The NeuroML support in neuroConstruct has been extended greatly and now all of the core model components in a project can be exported in a Level 3 NeuroML file and reloaded with little loss of information (section 3.5.4). Support for abstract cell models in NeuroML v2.0 has also been added (section 3.6).
- **Automated testing of neuroConstruct:** Incorporation of unit testing has improved the stability of the core of neuroConstruct and facilitated automated testing of simulator code generation and model behaviour across simulators (section 4.5).

2.3 Interaction with other software packages

Close interaction with other widely used applications in computational neuroscience is an important part of neuroConstruct's functionality. How the application interfaces with a number of neuronal simulators and general data analysis applications is outlined here.

2.3.1 Simulator support

Simulators have different features and focus on different types of models. The ability of neuroConstruct to generate scripts to run a particular model on a simulator depends on all features of that model being supported by the simulator and whether there is a mapping from NeuroML to the corresponding entity in the simulator's native format. An overview of which features of NeuroML are supported by simulators is given in Table 3.1.

NEURON

Generation of NEURON code was one of the first features of neuroConstruct. The generated network is converted to NEURON's native hoc format for the cells, network connectivity and simulation settings, and NMODL scripts (mod files) are generated and automatically compiled for the channels and synapses (Carnevale and Hines (2006) contains full details of the hoc and NMODL languages).

As for all simulators, NEURON is executed as a separate process, and there is no interaction with neuroConstruct during the simulation run. After the simulation run is finished, the user can interact with the created cells and network as normal through the NEURON command line interface. Blocks of custom NEURON code can be specified through the neuroConstruct GUI for insertion at specified times in the execution cycle (e.g. after cell creation, just before or after main simulation executes), allowing great flexibility, though in a simulator dependent way.

There is also an option for generating NEURON simulation code for that platform's Python interface. In this case the network structure is specified in a NetworkML file (XML or HDF5) resulting in much smaller total size for simulation scripts (section 4.2.3). Simulations in Parallel NEURON are also now supported (section 4.2.1).

GENESIS and related simulators

GENESIS 2 has also been a core target simulator for neuroConstruct for many years. The majority of detailed cell and network models developed on neuroConstruct run successfully on this platform (chapter 5), provided there are supported mapping of the channel and synapse types from ChannelML. While GENESIS itself is in theory extensible (by recompiling the core with new C++ classes for new channel and synapse models), the focus has been on models which could run on the standard GENESIS distribution. Simulations can be generated in either SI or physiological units (Table 2.1).

MOOSE supports the majority of GENESIS 2 objects and almost all cell models running on GENESIS will run on this platform too. neuroConstruct includes a checkbox at the Export GENESIS tab to customise scripts for MOOSE and execute them in the same way.

Initial testing with GENESIS 3/Neurospaces has shown that some simple neuroConstruct generated GENESIS 2 scripts will run on this platform, but the small user base for this platform has meant that greater support for this platform has not yet been a priority.

PSICS

This simulator focussed on detailed single cell modelling and does not yet support networks. Single cells with current clamp inputs generated in neuroConstruct can be run on this with some restrictions on the channel models supported (e.g. PSICS does not yet support complex intracellular Ca^{2+} dynamics needed for BK and SK channels).

PSICS uses its own algorithm for spatiality discretisation of complex morphologies[12] and the main parameter for this, specifying the maximum dimension of the sections which the dendritic tree should be split up into, can be set through the neuroConstruct GUI. The other parameter to set is the default single ion channel conductance. As channel distributions are specified as densities in neuroConstruct/NeuroML, a value for this is needed to calculate the individual channel numbers for PSICS.

[12]http://www.psics.org/guide/process

PyNN

A number of neuronal network simulators have recently introduced scripting interfaces based on the Python scripting language. The PyNN initiative (Davison et al, 2008), which started as part of the EU FACETS project, seeks to create a specification for a common set of Python commands for setting up neuronal network simulations. Simulators which currently support the language include NEURON, NEST, Brian, MOOSE and PCSIM[13] and there is also work to support the running of such networks on VLSI neuromorphic hardware created by the FACETS project.

The focus of the PyNN to date has been on large scale networks of simple spiking neurons. For this reason, the only types of models which can currently be exported to this format are networks of simple Integrate and Fire networks connected with fixed or plastic synapses. Nevertheless, good agreement in model behaviour across simulators like NEST and NEURON is possible (Gleeson et al, 2010a), laying a solid basis for the greater interoperability between PyNN and NeuroML v2.0 (section 3.6).

2.3.2 Other tools and languages

MATLAB/GNU Octave

While neuroConstruct's built in network analysis tools facilitate interactive investigation of network behaviour, complex post processing often needs to be done in a dedicated data analysis package like MATLAB (MathWorks). An option in the main neuroConstruct settings enables generation of a small set of MATLAB files for each simulation which facilitates loading data into this package. GNU Octave[14] is a free, open source alternative to MATLAB

[13]http://www.lsm.tugraz.at/pcsim

[14]http://www.gnu.org/software/octave

and can also be used for these generated files.

Igor Pro

In a similar manner as for MATLAB, scripts can be generated for loading simulation data into Igor Pro (WaveMetrics). This facilitates comparison of computational data with electrophysiological data, for example using the free NeuroMatic set of functions[15].

HDF5

neuroConstruct uses HDF5 as a compact binary format for saving and reloading generated networks in NetworkML and as an optional format for simulators to save voltage traces or spike times (section 4.3). Traces of experimental or simulation data from other sources[16] can be loaded into neuroConstruct, which does not need to know the exact structure of the data but will parse the contents and list all appropriate data sets which can be plotted.

2.4 Using neuroConstruct

2.4.1 Installing and running neuroConstruct

neuroConstruct, together with a number of example projects, can be freely downloaded from http://www.neuroconstruct.org/download. Automatic installers are provided for Windows, Linux and Mac OS. A zipped file is also available which includes the compiled jar file and which can be used on any of these platforms.

[15]http://www.neuromatic.thinkrandom.com

[16]For example, NeuroTools http://neuralensemble.org/trac/NeuroTools

The software is developed under the GNU General Public Licence and the source code for neuroConstruct is included with the standard download. The download page also gives details of how to access the Subversion repository which hosts the latest code.

The application can be launched in a number of ways, including (if one of the automatic installers above is used for install) via a desktop link or entry in Start/Application menu. Manual execution of the application in a command terminal is possible with the ***nC*** utilities:

Linux/Mac:	***./nC.sh myProject.ncx***
Windows:	***nC.bat myProject.ncx***

Apache Ant[17] can also be used for running neuroConstruct (***ant run***), and also for recompiling and testing the code (section 4.5).

2.4.2 Documentation

Detailed user and developer documentation for neuroConstruct is available online at http://www.neuroconstruct.org/docs. This includes full installation instructions, a number of tutorials to facilitate learning to use neuroConstruct, and a glossary of terms used in the application. There are also more details of the interactions with NeuroML, the Python interface and the Java API.

2.5 Conclusions

The ability to position and connect complex 3D network models in neuroConstruct, and automatically generate code for multiple widely used sim-

[17]http://ant.apache.org

ulators represents an important advance in making anatomically realistic network models more accessible for a wider range of researchers.

Cell models can be created through the GUI which incorporate detailed reconstructed morphologies, and a range of voltage and ligand gated conductances distributed across their membranes. Networks can be created based on experimental data on synaptic innervation domains, can incorporate heterogeneous connectivity properties, and use a variety of realistic synaptic mechanisms. Multiple instances of cell and network models can be generated, simulated and managed, to investigate the behaviour of populations of models.

Automated code generation for simulators such as NEURON, GENESIS and MOOSE is a unique feature of neuroConstruct and lowers the barriers to the use of detailed neuronal models for non-computational neuroscientists, and also those outside the field who want to learn about the properties of anatomically and biophysically realistic neuronal systems.

Chapter 3

NeuroML

This chapter outlines the motivation for standardisation in computational neuroscience, provides a brief history of the development of NeuroML, describes the current version of the language of which I was the main technical contributor and outlines the resources I have developed for researchers who wish to use NeuroML in their modelling work. It also describes the requirements for and initial implementation of NeuroML v2.0.

3.1 The need for standardisation in computational neuroscience

Development of a computational model of part of the nervous system is essentially the encapsulation of a conceptual model describing the structure of the system and how its elements interact over time. Such models often incorporate new measurements on the biophysical properties of its components or aim to reproduce new experimental data on the behaviour of the system. Sharing such models allows other researchers to critique the underlying assumptions and reuse the anatomical or physiological data contained in the model. Often however, there is general agreement of the core formalisms used for describing such systems, and a number of such conventions are widely used in computational neuroscience, e.g. the Hodgkin Huxley model, cable theory, compartmental representations of neuronal structure. Exchange of ideas in the form of models should therefore rely less on redefining these conventions, and more on the novel physiological data which is contained in them.

A standardised language for exchanging computational neuroscience models would facilitate this exchange of data and ideas. New cell or channel models for example would only contain the relevant physiological properties of the model, structured according to commonly used formalisms. Well structured, machine readable definitions of models would also greatly facilitate software interoperability in the field. This has been successfully practised for years in the Systems Biology community when exchanging models of signalling pathways, though the range of entities important for information processing in the nervous system has meant that coming up with working standards for neuronal models has been difficult until recently.

3.2 The development of NeuroML

3.2.1 Pre 2004

A series of meetings around 2000 attended by parties interested in developing a common language for specifying computational neuroscience models in XML (Extensible Markup Language, (Bray et al, 1998)) led to a publication describing the initial aims of the NeuroML initiative (Goddard et al, 2001). A number of these goals were in line with earlier work in this area (Gardner et al, 2001). This early effort provided a set of templates for describing neuronal models at the channel, cellular and network levels, which subsequently led to software implementations by some of the original NeuroML contributors, including NeoSim (Howell et al, 2003) and the NeuroML Development

Kit (NDK). While several other software projects adopted these templates and used the NDK (for example KInNeSS[1], Virtual RatBrain[2]), adoption of this version of the NeuroML language was lower than expected in the growing computational neuroscience software development community. Some of the potential reasons for this were lack of communication between the various simulator communities, the relatively small number of detailed models which were publicly available at the time, and other, more pressing priorities among simulator developers, for example improving simulator performance and adding new features including graphical interfaces.

3.2.2 NeuroML version 1.x

A new approach for the NeuroML initiative was adopted following discussions in 2004 and 2005 at the GENESIS and NEURON user group workshops. At this time a new language for describing neuronal morphologies in XML (MorphML) was under development (Crook et al, 2007; Qi and Crook, 2004). Independently, I was developing neuroConstruct (chapter 2), at the time focussed on generating neuronal simulations for the NEURON and GENESIS simulators. neuroConstruct had its own internal representation for morphologies, channel and networks which could be mapped to multiple simulator specific formats. It was agreed that these efforts should be merged under the banner of NeuroML, and the current structure of the NeuroML language (referred to as v1.x) was created. This new structure was split into three Levels and incorporated MorphML, ChannelML and NetworkML (section 3.3.2), providing greater modularity of the language and giving application developers the freedom to choose to support only part of the language as needed.

[1]http://symphony.bu.edu
[2]http://www.virtualratbrain.org

This modular approach focused on the elements of biophysically detailed neuronal models which can be analysed in isolation and are targets for reuse between models. The description of a typical biophysically detailed model includes: the structure of a neuron's dendritic/axonal arborisation in 3D (its morphology); the distribution of ion channels across this morphology; the kinetics of these ion channels; the properties of the synaptic mechanisms associated with the neuron; and the 3D cell positions and connectivity patterns of complex networks.

The initial priority for this version of NeuroML (Gleeson et al, 2010a) was to create a format for expressing these core model elements in a language which could be mapped to the most widely used neuronal simulators at that time, NEURON (Carnevale and Hines, 2006) and GENESIS (Bower and Beeman, 1997), although a number of other simulators and visualisation tools have since added NeuroML support (section 3.5). This resulted in a language focused on conductance based (multi-) compartmental models of neurons (types II-IV in Figure 1.1B), although it includes support for some basic types of abstract neuronal models (type V in Figure 1.1B). The specification of simulation parameters (run time, integration method, etc.) was not included, as the language was designed to specify the models themselves as opposed to the details of how the simulations were run[3].

NeuroML v1.x developed over a number of iterations through actively converting published models to the format, testing them across multiple simulators and engaging with the simulator development community. It has been stable in its current form (v1.8.1) for the past 2 years.

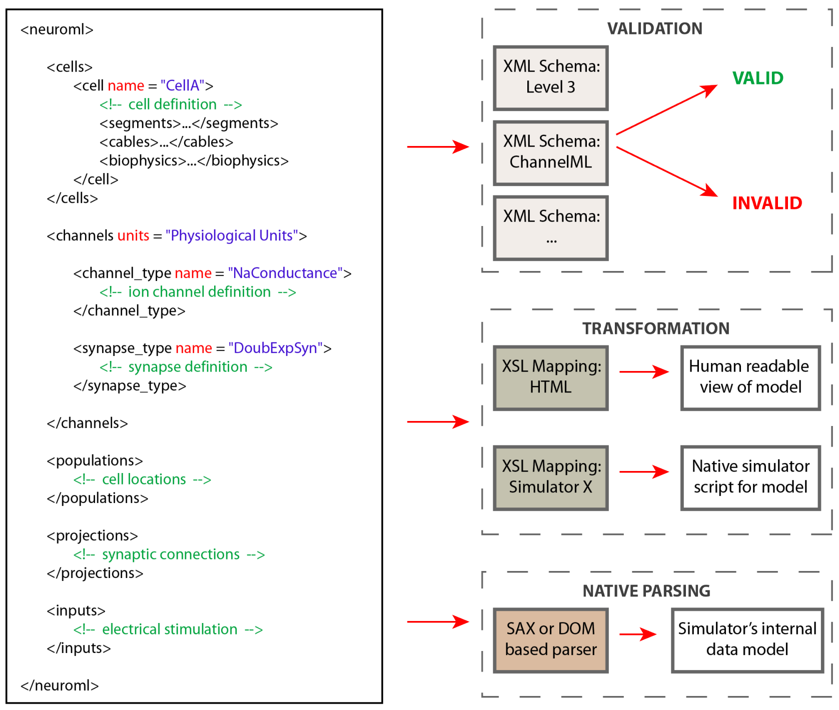
[3]SED-ML (Simulation Experiment Description Markup Language, http://www.sed-ml.org) is a language designed for encoding details of simulation setup and execution and is increasingly used in the SBML and CellML communities. It is appropriate for describing NeuroML simulations also.

3.3 Structure of NeuroML version 1.x

3.3.1 Technical approach to language specification

A NeuroML document consists of XML elements describing the physiological components of the neuronal model. Figure 3.1 shows an example of a NeuroML file containing various elements such as ***cells***, ***channels*** and ***populations***. The structure of a valid NeuroML document is defined using XML Schema Definition (XSD) files. Using these, standard XML handling libraries can be used to check the validity of an XML file against the various modules of the language. An error will be generated if, for example, the ***name*** attribute is missing from the ***cell*** element. The XML Schema files used for the language are discussed further in section 3.3.3.

Once an XML file is known to be valid NeuroML, the contents of the file can be transformed into other formats in a number of different ways. An application can read the XML natively using one of the commonly used parsing frameworks such as SAX (Simple API for XML) or DOM (Document Object Model). An alternative approach is to transform the XML description into another text format which can be natively read by an application. This is possible with Extensible Stylesheet Language (XSL) files. Examples of these files are available for mapping NeuroML files onto HTML format (for making a more human readable web page description of the model), and a number of simulators' own script formats, including NEURON, GENESIS and PSICS. This approach has the advantage that applications need not be reimplemented to natively support NeuroML, but can still have access to models in the format.



```
<neuroml>

    <cells>
        <cell name = "CellA">
            <!-- cell definition -->
            <segments>...</segments>
            <cables>...</cables>
            <biophysics>...</biophysics>
        </cell>
    </cells>

    <channels units = "Physiological Units">

        <channel_type name = "NaConductance">
            <!-- ion channel definition -->
        </channel_type>

        <synapse_type name = "DoubExpSyn">
            <!-- synapse definition -->
        </synapse_type>

    </channels>

    <populations>
        <!-- cell locations -->
    </populations>

    <projections>
        <!-- synaptic connections -->
    </projections>

    <inputs>
        <!-- electrical stimulation -->
    </inputs>

</neuroml>
```

Figure 3.1: Example NeuroML file containing cells, channels, synapses and network elements. Some elements omitted for clarity. The validity of the file can automatically be checked against the NeuroML specification contained in XML Schema Description (XSD) files (section 3.3.3) . Extensible Stylesheet Language (XSL) files can be used to transform the contents of the file into formats which can be read more easily by humans or by other applications (for example a simulator's native file format). Alternatively, an application can parse the XML using one of the standard XML parsing techniques and transform the contents into its own data representation. Note that a complex NeuroML model such as this could also be split between individual files for each cell, channel, synapse and one for the network structure.

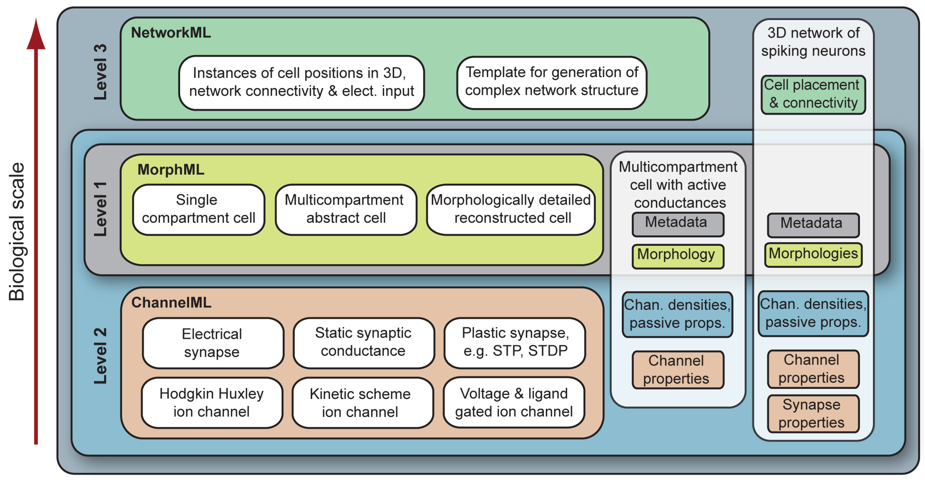

Biological scale
Level 3
NetworkML
Instances of cell positions in 3D, network connectivity & elect. input
Template for generation of complex network structure
3D network of spiking neurons
Cell placement & connectivity
Level 1
MorphML
Single compartment cell
Multicompartment abstract cell
Morphologically detailed reconstructed cell
Multicompartment cell with active conductances
Metadata
Morphology
Metadata
Morphologies
Level 2
ChannelML
Electrical synapse
Static synaptic conductance
Plastic synapse, e.g. STP, STDP
Hodgkin Huxley ion channel
Kinetic scheme ion channel
Voltage & ligand gated ion channel
Chan. densities, passive props.
Channel properties
Chan. densities, passive props.
Channel properties
Synapse properties

Figure 3.2: The three Levels as used in NeuroML v1.x, and the subcomponents, MorphML, ChannelML and NetworkML. Examples of models which can be created within and across the various parts of the language are given in the white rounded boxes. Reproduced from Gleeson et al (2010a).

3.3.2 Levels in NeuroML version 1.x

Three Levels are defined in NeuroML v1.x to facilitate modular use of the language (Figure 3.2). These are related to the different biological scales present in the systems being modelled. As models of single neurons are essential to most of the systems to be described, specifications for the structure of individual cells form the core of Level 1. Level 2 builds on this by including definitions of the electrical properties of these cells, allowing for specifications of spiking cell models. Level 3 is used for networks of these cell models in 3D.

Level 1: Morphologies and metadata

The first Level of the NeuroML language has two main purposes: to define neuronal morphologies (MorphML) and metadata, which provides additional information about model components at this and subsequent levels. The elements permitted for a cell description at Level 1 and subsequent Levels is shown in Figure 3.3 (a detailed description of each of these elements is given in Supporting Text S1 of Gleeson et al (2010a)). Cells are specified in the ***cell*** element and are described by lists of ***segment*** elements, with each containing the 3D location and diameter of its ***proximal***[4] and ***distal*** ends. The ***cable*** element is used for unbranched sets of ***segment*** elements and can be used as the basis for specifying regions of interest on the cell (e.g. apical or basal dendrites). Details of the mapping between elements in MorphML and the data structures of other applications that use morphology formats such as Neurolucida, NEURON and GENESIS are described in Crook et al (2007) and section 2.2.1. MorphML also allows description of other anatomical information, which may have been recorded during cell reconstruction, such as histological features, reference points, and outlines of perceived boundaries (Crook et al, 2007).

NeuroML Level 1 also allows specification of metadata, which is important for tracking the provenance of the model components and for providing background information on the model. A number of elements are included to provide structured information on the original authors of the model and translators of the model to NeuroML format (***authorList*** element), publications associated with the model (***publication***), and references to entries in databases such as ModelDB (***modelDBref***, Hines et al (2004)) and NeuroMorpho.org (***neuroMorphoRef***, Ascoli et al (2007)), as well as more general text based comments (***notes***) and other semi structured metadata (***properties***, ***annotation***). The concept of model stability (***status***) is also included to allow a record of any known limitations of the model. Two types of unit system are allowed in NeuroML: SI units and physiological units (ms,

[4]The ***proximal*** element can be omitted if it is the same as the parent's ***distal*** point

mV, cm, etc., same as GENESIS physiological units in Table 2.1), and only one of these must be used consistently in relevant elements of a NeuroML file. This facilitates the correct conversion of physical quantities to the unit system of each supported application. Note that lengths, e.g. the 3D coordinates and diameter of a ***segment***, are usually expressed in micrometers, as this is the most widely used unit for such values.

Level 2: Channels, synapses and channel distribution

The second Level of the NeuroML language describes the electrical properties of the membrane that underlie rapid signalling in the brain. The two main parts of this Level are: an extension of the morphological descriptions from Level 1 that includes details of the passive electrical properties and channel densities on various parts of the cell (Level 2 cell in Figure 3.3); and ChannelML, which allows descriptions of the individual conductance mechanisms (Figure 3.4). ChannelML supports two main types of conductances: those that arise from channels distributed over the plasma membrane (***channel_type*** element, for example see Figure 3.5), such as voltage-gated conductances or conductances gated by intracellular ions (for example [Ca^{2+}] dependent potassium conductances); and conductances arising at synaptic contacts (***synapse_type***). Distributed conductances are normally specified by describing the transition rates between channel states and their voltage dependence (Figure 3.5). This allows specification of channel gating models with the traditional Hodgkin-Huxley formalism (with multiple instances of identical gates) or with more detailed state-based kinetic (Markov) models (of which the HH model is a special case).

A wide range of examples of voltage-gated conductances are supported by ChannelML including those underlying fast and persistent Na^{+} currents,

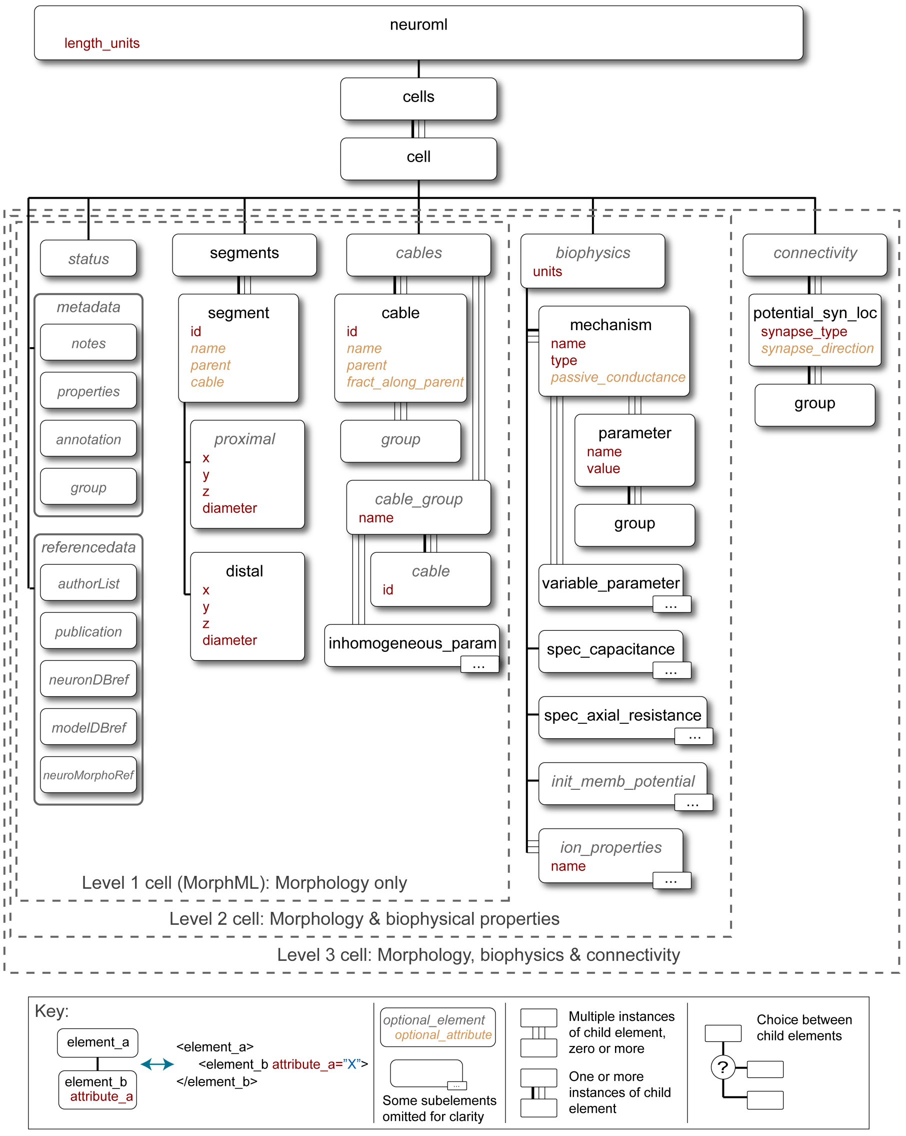


Figure 3.3: Elements for representing cells in NeuroML Levels 1-3. The main element for expressing a branching neuronal structure is ***cell*** which is used for all Levels in NeuroML. The core of this is a set of ***segment*** elements which describe the 3D shape of the cell. These can be grouped into ***cables*** which represent unbranched neurites of the cell. Metadata present in the cell description can contain details of the creators of the cell model, or the data on which it was based. Addition of the ***biophysics*** element allows a Level 2 conductance based spiking cell model to be described, and the ***connectivity*** element can be used for the allowed synaptic connectivity of a Level 3 cell. Reproduced from Supporting Text S1 of Gleeson et al (2010a).

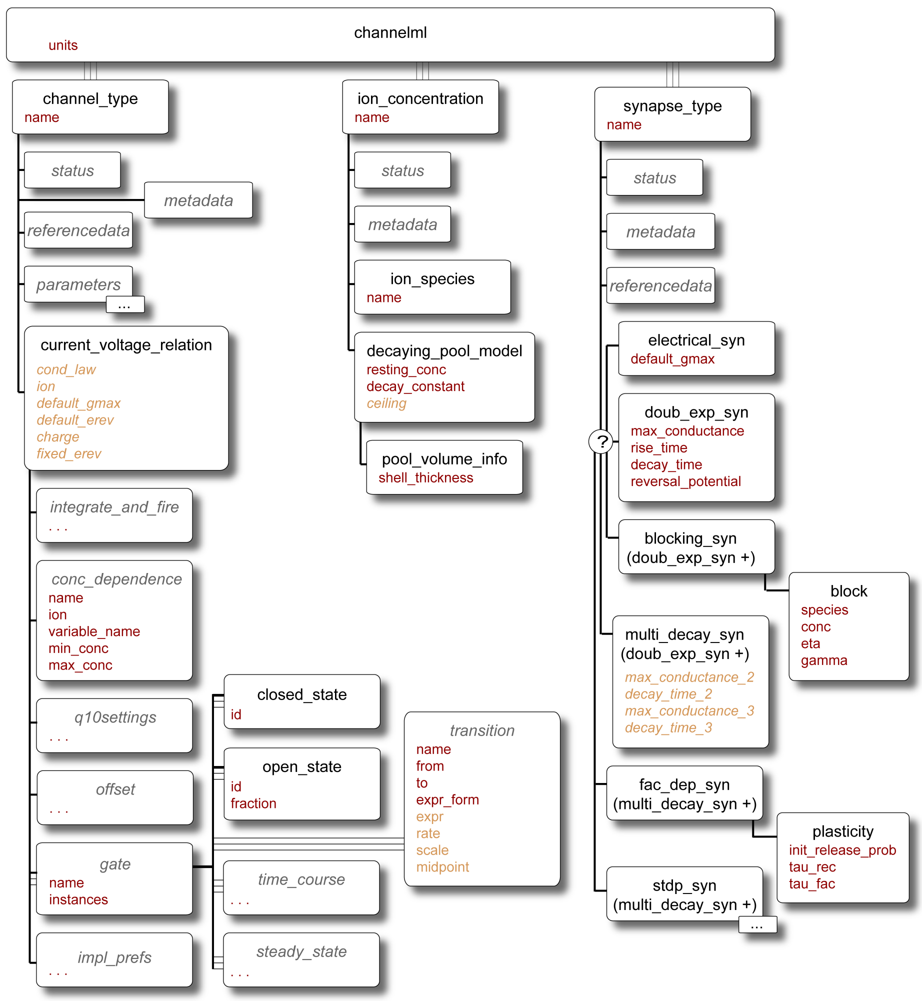
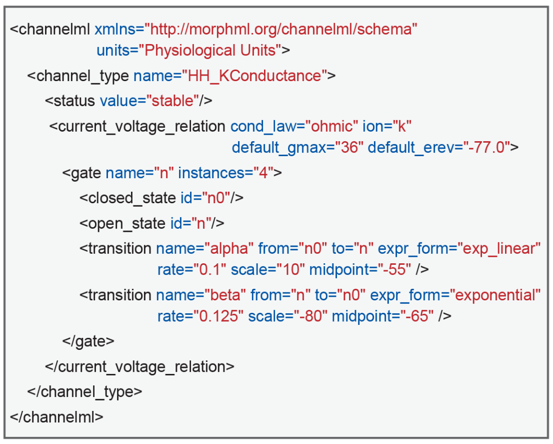


Figure 3.4: Elements in ChannelML. ChannelML allows expression of models of voltage (and ligand) gated conductances which are dispersed across the cell membrane (in ***channel_type*** element), conductances which are concentrated at synaptic contacts (in ***synapse_type*** element) and basic models of time varying internal ion concentrations (in ***ion_concentration*** element). Distributed conductance descriptions can contain a number of ***gate*** elements, which describe the transitions between conducting and non conducting states of the channels underlying the conductances. A number of synaptic conductance models are allowed including simple double exponential waveforms, AMPA and NMDA receptor mediated synapses, Short Term Plasticity (STP) models, Spike Timing Dependent Plasticity (STDP) models, and electrical synapses. The ***ion_concentration*** element can be used for the simple models of exponentially decaying Ca^{2+} pools often used in detailed cell models. Reproduced from Supporting Text S1 of Gleeson et al (2010a).

delayed rectifier, A- and M-type K^+ currents, H-currents and L- and T-type Ca^{2+} currents. $[Ca^{2+}]$ dependent BK and SK type channels can also be expressed. The commonly used Q10 function for temperature dependence of transition rates can be added (***q10settings*** element). While the focus of NeuroML to date has been on more detailed conductance based models, ChannelML also supports a basic Integrate and Fire neuron model (***integrate_and_fire***). However, more advanced types of reduced model such as exponential Integrate and Fire or Izhikevich spiking neurons are not supported in this version (see section 3.6 for support of more abstract neuronal representations in NeuroML v2.0).

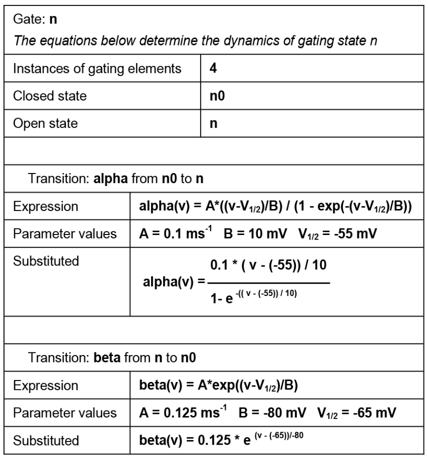
Both neurotransmitter gated conductances at chemical synapses and gap junction conductances at electrical synapses are supported in ChannelML (Figure 3.4). Conductance changes at chemical synapses are defined by a time course which can have a number of forms including an exponential rise and up to three decay components. These conductances include both the simple linear ohmic type (for modelling most AMPA (α-amino-3-hydroxy-5-methyl-4-isoxazolepropionic acid) and $GABA_A$ (γ-aminobutyric acid, class A) receptor mediated synapses) and nonlinear voltage-dependent components (e.g. for modelling the Mg^{2+} block of the NMDA (N-Methyl-D-aspartate) receptor mediated synaptic component). Activity dependent synaptic plasticity is implemented with two mechanisms in ChannelML: a short-term plasticity (STP) mechanism based on a widely used STP model (Tsodyks and Markram, 1997) incorporating both depression and facilitation components and a spike timing dependent plasticity (STDP) mechanism based on the model of Song et al (2000). NeuroML provides representations of phenomenological models of synaptic plasticity that can reproduce a wide range of behavior including short-term facilitation and depression and

A



```
<channelml xmlns="http://morphml.org/channelml/schema"
            units="Physiological Units">
   <channel_type name="HH_KConductance">
      <status value="stable"/>
       <current_voltage_relation cond_law="ohmic" ion="k"
                                default_gmax="36" default_erev="-77.0">
         <gate name="n" instances="4">
            <closed_state id="n0"/>
            <open_state id="n"/>
            <transition name="alpha" from="n0" to="n" expr_form="exp_linear"
                        rate="0.1" scale="10" midpoint="-55" />
            <transition name="beta" from="n" to="n0" expr_form="exponential"
                        rate="0.125" scale="-80" midpoint="-65" />
         </gate>
      </current_voltage_relation>
   </channel_type>
</channelml>
```

B



Gate: **n** *The equations below determine the dynamics of gating state n*	
Instances of gating elements	**4**
Closed state	**n0**
Open state	**n**
Transition: **alpha** from **n0** to **n**	
Expression	**$\text{alpha}(v) = A*((v-V_{1/2})/B) / (1 - \exp(-(v-V_{1/2})/B))$**
Parameter values	**$A = 0.1\ \text{ms}^{-1}$ $B = 10\ \text{mV}$ $V_{1/2} = -55\ \text{mV}$**
Substituted	**$\text{alpha}(v) = \dfrac{0.1 * (v - (-55)) / 10}{1- e^{-((v - (-55)) / 10)}}$**
Transition: **beta** from **n** to **n0**	
Expression	**$\text{beta}(v) = A*\exp((v-V_{1/2})/B)$**
Parameter values	**$A = 0.125\ \text{ms}^{-1}$ $B = -80\ \text{mV}$ $V_{1/2} = -65\ \text{mV}$**
Substituted	**$\text{beta}(v) = 0.125 * e^{(v - (-65))/-80}$**

C

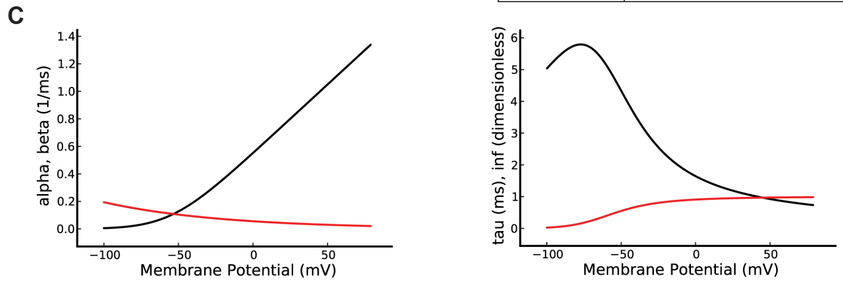


Figure 3.5: A) A ChannelML file containing a Hodgkin-Huxley type K^+ conductance model, with four instances of a gating mechanism with open and closed states, and the rates of transitions between them. Supporting Text S1 of Gleeson et al (2010a) contains a description of each of the elements contained in this file, and section 10.2 of that document outlines in more detail the equations behind a channel model expressed in ChannelML. B) A section of a HTML page automatically generated from the ChannelML using an XML Stylesheet (XSL) file. C) Top: plots of the forward (alpha, black) and reverse (beta, red) transition rates. Bottom: the time constant (tau = 1/(alpha + beta)) of the transition (black) and steady state of the gating variable (inf = alpha/(alpha + beta), red). These views can be generated automatically (e.g. by neuroConstruct) for any ChannelML valid file. Reproduced from Gleeson et al (2010a).

Hebbian and anti-Hebbian learning, thus accommodating synaptic plasticity over a wide range of time scales where adequate simulator support exists.

While ChannelML supports many of the channel and synapse model types which are used in published models today, they are essentially "hard coded" into the language, i.e. introducing a new synaptic plasticity model would involve updating the core of the language. Also, definitions of the behaviour of model elements are only present in textual descriptions in the specifications. These issues have been addressed with v2.0 of the language, allowing greater extensibility of the core of the language and machine readable definitions of model component dynamics (section 3.6).

Level 2 also allows the location and density of membrane conductances described in ChannelML to be specified on regions of the cell (e.g. soma, axon, apical dendrites). The passive electrical properties of the cell (e.g. specific axial resistance and specific membrane capacitance) can be defined in a similar manner (using the ***biophysics*** element; Figure 3.3). Moreover, non-uniform channel densities can be implemented using a metric, such as the path distance from soma, and expressing the density in terms of this metric (using the ***variable_parameter*** element). Although NeuroML Level 2 is required for defining a full spiking neuron model, elements of the models can be specified in standalone files containing MorphML and ChannelML, thereby facilitating the exchange and reuse of individual model components.

Level 3: Network connectivity

The third Level of NeuroML allows specification of the 3D anatomical structure and synaptic connectivity of a network of neurons, together with the properties of the external input used to drive the network. NeuroML Level 3 has two main purposes: to define NetworkML (Figure 3.6) and to allow extension of Level 2 cells with specification of regions of the cell membrane (e.g. apical dendrites) to which specific synaptic connections are limited (***connectivity*** element; Figure 3.3). Thus complex networks with different types of excitatory and inhibitory neurons can be defined, including dendritic sub-region specific synaptic connections. There are two possible ways to describe networks in NetworkML: an explicit list of instances of cell positions and synaptic connections (instance based representation); or as an algorithmic template for describing how instances of the network should be generated, for example to place 300 cells randomly in a certain 3D region (template based representation). The instance based representation is quite compact, even for large scale simulations, because a network with 10,000 identical neurons will only have one instance of the cell description and a list of 10,000 locations. To date, this has proven a more useful and portable format than the template based representation. Only a limited range of network templates is currently supported, though these are in the process of being updated for the next version of NeuroML (see section 3.6). The instance based representation can also include information on the computational node a cell should be run on (***node_id*** attribute) to facilitate execution of large scale networks on parallel computing hardware.

There are three core elements for describing networks in NetworkML: ***population*** specifies the numbers of cells of a specific type, together with their locations in 3D space; ***projection*** defines the set of synaptic connections between two populations or within a single ***population***, by identifying the precise location of the synapse on the pre- and postsynaptic neuronal morphology and specifying the type of synapse(s) present; and ***input*** describes an external electrical input into the network. Inputs can take the form of a current pulse delivered by model electrodes or random synaptic stimulation.

3.3.3 Schemas for Levels

The NeuroML specification is split into a number of XML Schema documents that define different aspects of the model description language. These provide explicit, machine readable definitions that specify, for example, that the ***distal*** element needs to contains attributes ***x***, ***y***, ***z*** and ***diameter***, all of which are double precision floating point numbers.

For Level 1, the file **MorphML_vXXX.xsd**, where the XXX is replaced by the current version number, defines the morphological elements, e.g. **Metadata_v1.8.1.xsd**. Metadata are defined in **Metadata_vXXX.xsd**. The Schema for MorphML imports the Schema for Metadata, so elements of Metadata will be in the namespace associated with that Schema, for example ***meta:notes***. A file valid against **Metadata_vXXX.xsd** will have root element ***morphml***. Another schema file has been defined, **NeuroML_Level1_vXXX.xsd**, which imports both of these Schemas, and files valid against this will have root element ***neuroml***.

Level 2 adds a Schema for ChannelML (**ChannelML_vXXX.xsd**), which also imports Metadata and defines the structure of files having root ***channelml***. **Biophysics_vXXX.xsd** is not used for standalone files, but contains details of the elements for passive electrical properties and channel distributions of cells and is imported into **NeuroML_Level2_vXXX.xsd** (along with the Schemas for Metadata, MorphML and ChannelML) to define Level 2 cells (or a file containing cells and channel definitions) with root element ***neuroml***.

Level 3 deals with network descriptions and contains a Schema for standalone network descriptions (**NetworkML_vXXX.xsd**, with root element

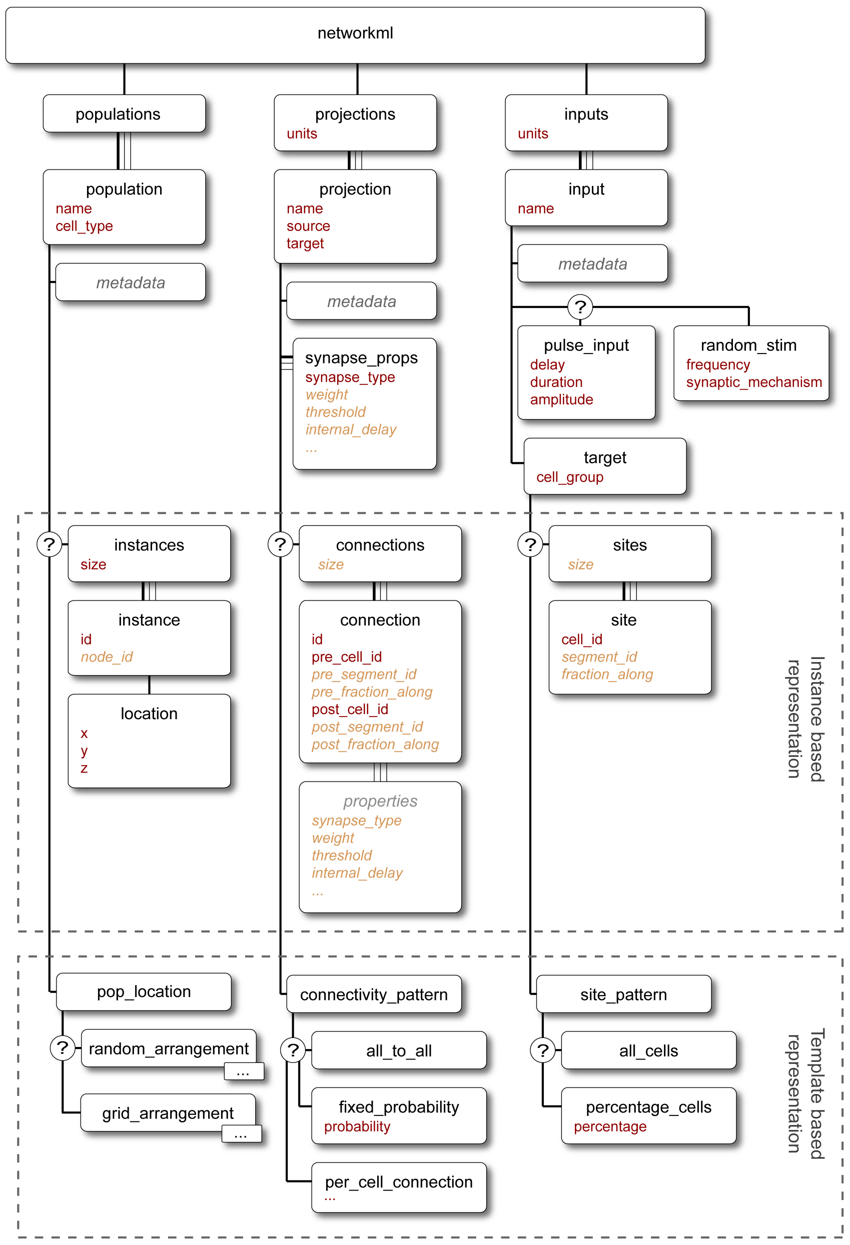
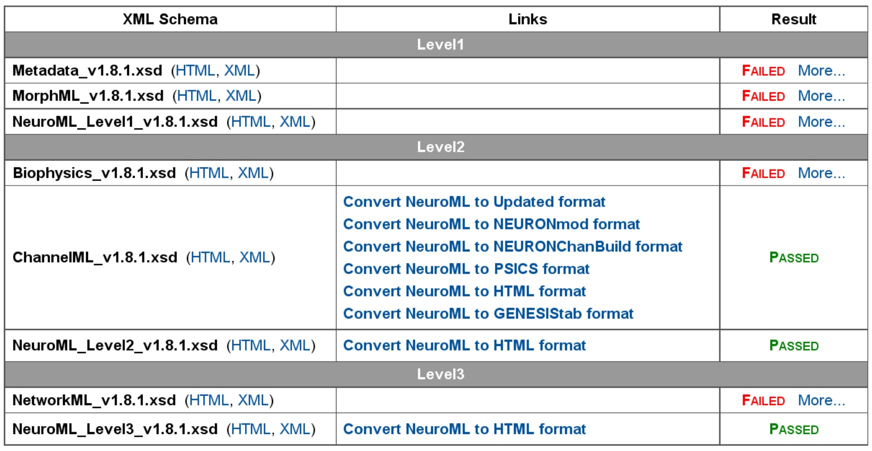


Figure 3.6: Elements in NetworkML. The core elements for expressing networks are: ***population*** for homogeneous groups of cells positioned in 3D; ***projection*** for synaptic contacts between (or within) populations and ***input*** for electrical stimulation to the network. The networks can either be expressed as lists of precise positions, connections and input locations (instance based representation) or as templates for generating these lists (template based representation). Reproduced from Supporting Text S1 of Gleeson et al (2010a).



XML Schema	Links	Result
Level1		
Metadata_v1.8.1.xsd (HTML, XML)		FAILED More...
MorphML_v1.8.1.xsd (HTML, XML)		FAILED More...
NeuroML_Level1_v1.8.1.xsd (HTML, XML)		FAILED More...
Level2		
Biophysics_v1.8.1.xsd (HTML, XML)		FAILED More...
ChannelML_v1.8.1.xsd (HTML, XML)	Convert NeuroML to Updated format Convert NeuroML to NEURONmod format Convert NeuroML to NEURONChanBuild format Convert NeuroML to PSICS format Convert NeuroML to HTML format Convert NeuroML to GENESIStab format	PASSED
NeuroML_Level2_v1.8.1.xsd (HTML, XML)	Convert NeuroML to HTML format	PASSED
Level3		
NetworkML_v1.8.1.xsd (HTML, XML)		FAILED More...
NeuroML_Level3_v1.8.1.xsd (HTML, XML)	Convert NeuroML to HTML format	PASSED

Figure 3.7: The validation of a ChannelML file containing a single channel model description using the online validator application[6]. As can be seen, the file is not valid against any of the Level 1 Schemas, is valid against the ChannelML and Level 2 Schemas, as well as the main Level 3 Schema (which includes ChannelML) but not NetworkML. Various options are provided for transforming the ChannelML into other formats (via XSL mapping files).

networkml), and one for Level 3 files (**NeuroML_Level3_vXXX.xsd**, root element ***neuroml***), one of which could contain all the elements for the cells, channels, synapses, positions and synaptic connections in a complex 3D network (as in the example in Figure 3.1).

Figure 3.7 shows a screenshot of the online validator application (see section 3.5.2) showing the results of the validation of a ChannelML file against these Schemas.

3.4 Organisational structure of NeuroML

The NeuroML language has been developed as a project on the SourceForge website[7] since the current modular structure took shape in 2005. The mailing lists available there are the main source of information on activities in

[7]http://sourceforge.net/projects/neuroml

the project. NeuroML development is based on an open, community driven process, and participation is actively encouraged from the neuronal modellers and application developers, and also from the wider computational biology and experimental neuroscience communities.

The organizational structure of the NeuroML initiative was given a more formal structure at the first NeuroML Development Workshop in March 2009 in London. The NeuroML Team was formed, which consisted of Robert Cannon (Textensor Limited), Sharon Crook (Arizona State University), Angus Silver (UCL) and me. The members of this team were responsible for the promotion of the initiative within the community, gaining funds for the continued work on the project and organising the annual workshops.

At the NeuroML Development Workshop in March 2011 in London, this core team was expanded into a 10 member Scientific Committee to drive forward the development of NeuroML. Initial membership of this consisted of the four NeuroML Team members along with Upi Bhalla (NCBS), Avrama Blackwell (George Mason University), Hugo Cornelis (K.U. Leuven), Andrew Davison (CNRS), Lyle Graham (Universit Paris Descartes) and Michael Hines (Yale University).

The International Neuroscience Coordinating Facility (INCF) was formed in 2004 through the Global Science Forum of the OECD with the aim of promoting international collaboration in the area of neuroinformatics. Its activities include a number of themed Programs which concentrate on areas of interest to the community, arrange meetings and develop standards and guidelines to facilitate collaborative research. One of the Programs which has been set up is on Multiscale Modelling of Neuronal Systems. Current work in this area involves development of NineML[8], a layered language for

[8]http://software.incf.org/software/nineml

describing large scale models of spiking neurons. NeuroML Team members are present on the Oversight Committee of this Program and I have been part of the Task Force ensuring developments in NeuroML v2.0 will be closely aligned with progress in this initiative (section 7.2.1).

3.5 Tool support for NeuroML

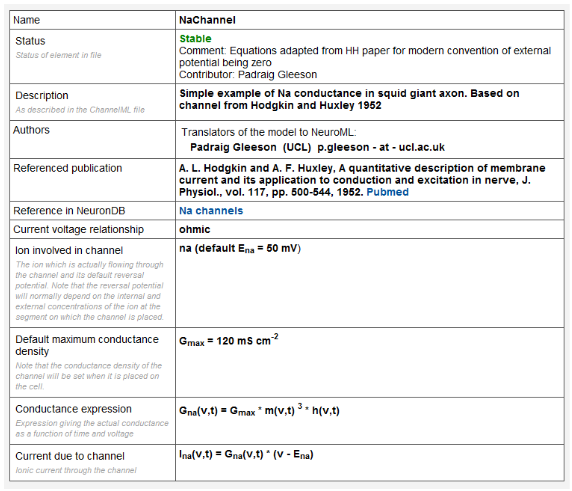
As discussed, the development of the NeuroML language went hand in hand with the conversion of published models to the format and development of support in existing simulators for the language. This section discusses a number of the tools and resources which I have played a significant part in developing as part of this work. As with the implementation of neuroConstruct, these tools have been made open source, allowing any interested party to contribute to the development. Significant contribution by other parties to the tools mentioned below are noted. An updated list of all software applications with NeuroML support is being maintained at http://www.neuroml.org/tool_support, and includes a number of other applications which have independently added NeuroML support.

3.5.1 NeuroML language specifications

The latest version of the NeuroML language specifications (v1.8.1) can be obtained online[9]. As outlined in section 3.3.1, the language is defined in a set of XML Stylesheet Document (XSD) files. These files can be downloaded individually, or can be viewed in a web browser converted to a more readable format. A detailed explanation of the elements allowed in NeuroML files at each Level is contained in Supporting Text 1 of Gleeson et al (2010a). These specification files, along with a history of previous versions, can also

[9]http://www.neuroml.org/specifications

A



Name	**NaChannel**
Status *Status of element in file*	**Stable** Comment: Equations adapted from HH paper for modern convention of external potential being zero Contributor: Padraig Gleeson
Description *As described in the ChannelML file*	**Simple example of Na conductance in squid giant axon. Based on channel from Hodgkin and Huxley 1952**
Authors	Translators of the model to NeuroML: **Padraig Gleeson (UCL) p.gleeson - at - ucl.ac.uk**
Referenced publication	**A. L. Hodgkin and A. F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, J. Physiol., vol. 117, pp. 500-544, 1952.** Pubmed
Reference in NeuronDB	**Na channels**
Current voltage relationship	**ohmic**
Ion involved in channel *The ion which is actually flowing through the channel and its default reversal potential. Note that the reversal potential will normally depend on the internal and external concentrations of the ion at the segment on which the channel is placed.*	**na (default E_{na} = 50 mV)**
Default maximum conductance density *Note that the conductance density of the channel will be set when it is placed on the cell.*	**G_{max} = 120 mS cm^{-2}**
Conductance expression *Expression giving the actual conductance as a function of time and voltage*	**$G_{na}(v,t) = G_{max} * m(v,t)^{3} * h(v,t)$**
Current due to channel *Ionic current through the channel*	**$I_{na}(v,t) = G_{na}(v,t) * (v - E_{na})$**

B

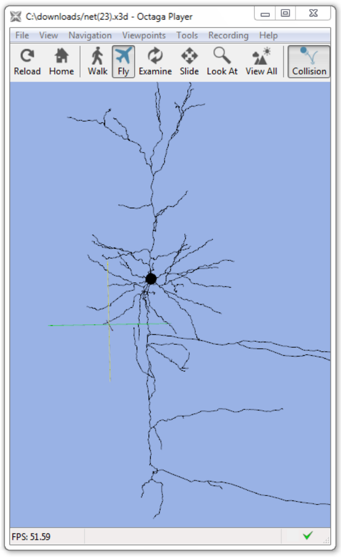


Figure 3.8: The NeuroML website has a number of features to convert NeuroML models into more accessible formats. A) Conversion of a ChannelML file into webpage summarising contents, offering links to other resources, etc. Only part of the model description is shown. B) A pyramidal cell morphology originally in MorphML was validated using the NeuroML Validator and then mapped to X3D format. It is visualised here using a web browser plugin (Octaga Player).

be obtained from the NeuroML version control repository on SourceForge (which uses Subversion[10]).

3.5.2 NeuroML Validator website

A useful tool for validating XML files against the NeuroML specifications is available at http://www.neuroml.org/NeuroMLValidator/Validation.jsp. The contents of a NeuroML file can be pasted into the text box provided and the application will validate the file against each of the NeuroML Schemas. An example of the results of the validation of a ChannelML file is given in Figure 3.7.

[10] http://neuroml.svn.sourceforge.net/viewvc/neuroml

Once a file has been successfully validated in this way, a number of options are given for converting the file to other formats using XSL mappings. For MorphML files, mappings to a HTML description of the cell or NEURON and GENESIS morphology files are provided. ChannelML files can be mapped to HTML (Figure 3.8A), NEURON (either NMODL or ChannelBuilder format as appropriate), GENESIS or PSICS. NetworkML files can be mapped to a description in HTML of the structure of the network.

Cell morphologies or instance based network descriptions can also be converted to X3D[11] format to visualize the structure of the cell or network in an X3D compatible browser plug-in. While this functionality is more limited than applications which read the NeuroML files natively and have inbuilt visualisation capability, it is useful for providing a quick 3D representation of the model (Figure 3.8B).

3.5.3 NeuroML example models

A number of published cell and network models have been converted to NeuroML format in the process of developing v1.x and many of these are available from http://www.neuroml.org/models. The majority of these are ones I have converted to NeuroML for validation of the language and testing of neuroConstruct, or as part of research collaborations with others in the Silver Lab. Figure 3.9 illustrates some of these examples. These are available as a zip file containing all the NeuroML files, or as a neuroConstruct project (section 2.1.3) which can be used to view the model and generate code for multiple simulators. Greater interaction with ModelDB (Hines et al, 2004) is in development to make it easier to search for models of specific cell types

[11] http://www.web3d.org/x3d

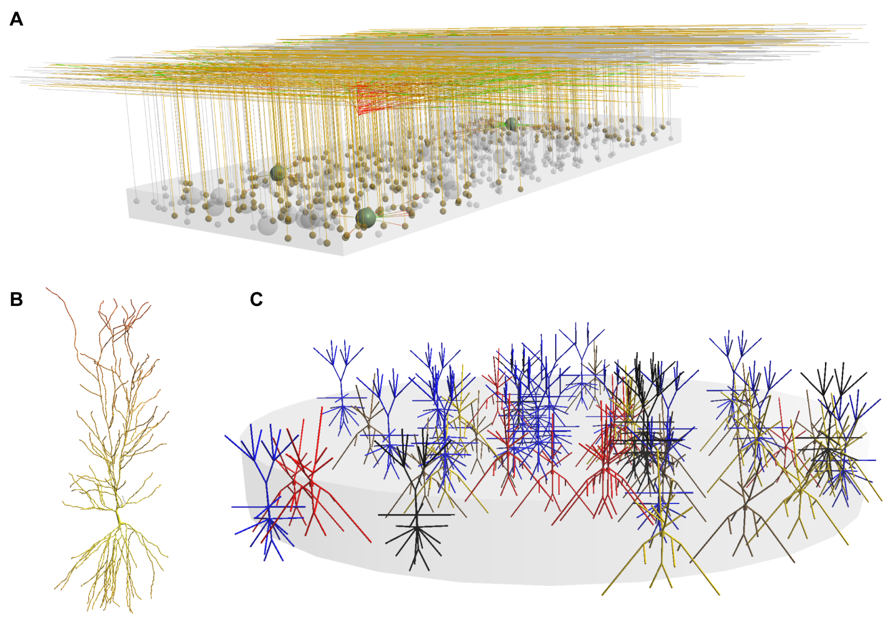


Figure 3.9: A number of models which are available in NeuroML. A) 3D network model of the granule cell layer of cerebellum (section 5.1.4). B) CA1 pyramidal cell model (section 5.16). C) Layer 2/3 cortical network model (section 5.2.4). Chapter 5 provides more details about each of these models.

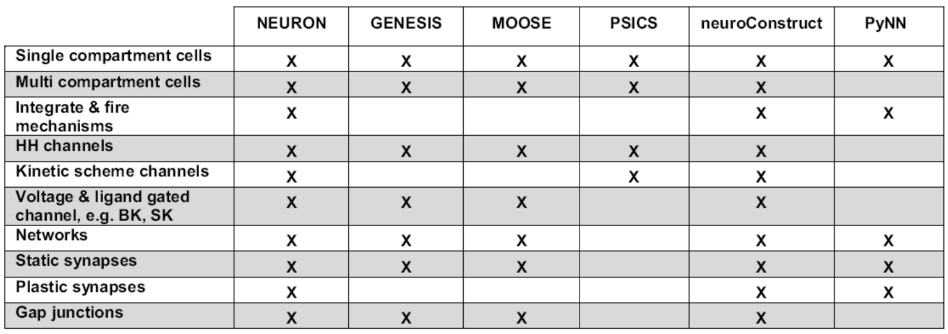
and brain regions[12]. A number of NeuroML models are also available at http://www.neuroConstruct.org/samples to illustrate the functionality of that application.

3.5.4 neuroConstruct

neuroConstruct (Gleeson et al (2007), chapter 2) is a graphical application to facilitate the development of networks of biophysically detailed neurons in 3D. Once cell and network models are created through the GUI, scripts to execute simulations of them can be generated for a number of applications including NEURON, GENESIS, MOOSE, PSICS and PyNN. neuroConstruct

[12]For example, http://senselab.med.yale.edu/modeldb/ShowModel.asp?model=127353 on ModelDB links to the neuroConstruct page for downloading the Traub et al (2005) thalamocortical network model

has been developed in parallel with the NeuroML specifications and there is native support for most parts of NeuroML. Table 3.1 provides a summary of the core features of NeuroML supported by neuroConstruct and a range of simulators.



	NEURON	GENESIS	MOOSE	PSICS	neuroConstruct	PyNN
Single compartment cells	X	X	X	X	X	X
Multi compartment cells	X	X	X	X	X	
Integrate & fire mechanisms	X				X	X
HH channels	X	X	X	X	X	
Kinetic scheme channels	X			X	X	
Voltage & ligand gated channel, e.g. BK, SK	X	X	X		X	
Networks	X	X	X		X	X
Static synapses	X	X	X		X	X
Plastic synapses	X				X	X
Gap junctions	X	X	X		X	

Table 3.1: Features of NeuroML supported by various applications. Different simulators focus on different types of modelling, and where a feature of NeuroML is supported by a simulator, a mapping for models of that type is available.

The internal representation of cells in neuroConstruct is closely based on MorphML (section 2.2.1). For historical reasons, cells in neuroConstruct consist of segments grouped in sections, whereas in MorphML the equivalent entities are segments grouped in cables. Most cells can be exported in NeuroML Level 3 and re-imported with no loss of information (including group information and non uniform channel distributions). Figure 3.10 shows the neuroConstruct interface with a cell visualised in 3D which can be exported/imported in NeuroML Level 1-3 formats.

neuroConstruct supports ChannelML descriptions of channel and synapse models. When generating the scripts for a particular simulator, neuroConstruct applies the corresponding XSL mapping to make a native representation of the model component and the generated file is compiled if necessary (as in the case of NEURON mod files). Cell mechanisms can also be included in neuroConstruct in the native simulator scripts (File Based Cell

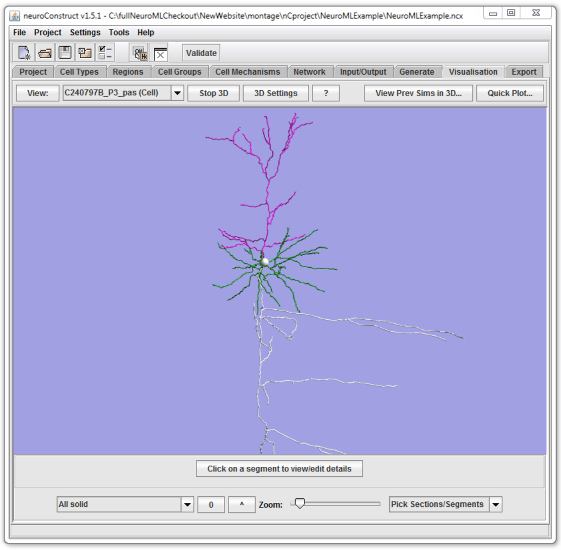

neuroConstruct v1.5.1 - C:\fullNeuroMLCheckout\NewWebsite\montage\nCproject\NeuroMLExample\NeuroMLExample.ncx
File Project Settings Tools Help
Validate
Project Cell Types Regions Cell Groups Cell Mechanisms Network Input/Output Generate Visualisation Export
View: C240797B_P3_pas (Cell) Stop 3D 3D Settings ? View Prev Sims in 3D... Quick Plot...
Click on a segment to view/edit details
All solid 0 ^ Zoom: Pick Sections/Segments

Figure 3.10: A pyramidal cell visualised in neuroConstruct from a MorphML cell description (same cell as Figure 3.8).

Mechanisms) and this can be useful in the process of converting a channel from one simulator's native format to ChannelML, as cells with two versions of a channel can be run side by side and compared directly[13]. There are a number of inbuilt features in neuroConstruct for generating plots from the contents of ChannelML files, for example of the voltage dependences of the steady-state activation and inactication variables and time constants or the synaptic conductance waveforms (Tab Cell Mechanism → (select a ChannelML based Cell Mechanism) → Edit selected Cell Mechanism → Generate associated plots).

NetworkML is used by neuroConstruct for storing and reloading the generated network structure (the instance based representation of network structure: explicit lists of cell positions and synaptic connections between identified points on pre- and post synaptic cells). When a network is generated (at tab Generate) it can be stored for future use in neuroConstruct or other application by pressing Save NetworkML. Options are present for storing in: XML text files (files can be viewed with text browser; produces large

[13]See http://www.neuroConstruct.org/docs/importneuron for more details

files; slower to generate and reload); zipped XML files (produces smaller files; slightly slower to save and reload); or HDF5 format (faster to save and reload; up to 90% smaller files; special software is needed to view and edit these types of files outside of neuroConstruct, for example HDFView[14].

There are also a number of options in neuroConstruct for exporting and importing NeuroML files combining elements from a number of different Levels. At tab Export → NeuroML, in addition to options for exporting only the cells in Levels 1, 2 or 3, the cells, channels, synapses and generated network structure can be exported in NeuroML, either as a set of separate files, or as a single NeuroML Level 3 file. When exported as one Level 3 file, there is an option to include annotations with neuroConstruct specific information (for example information on regions, cell group colors, plots, simulation configurations, etc.), which can be read when the Level 3 file is imported into a new, empty neuroConstruct project, facilitating model exchange between neuroConstruct users. The exported file is still in valid Level 3 format, and other NeuroML compliant applications can read the file, ignore the neuroConstruct specific annotations and just import the cells, channels, populations, etc.

A Level 3 file (generated by any NeuroML compliant application) containing a mixture of cells, channels and network information can be imported into neuroConstruct and a new project created, ready for export to supported simulators. This can be done through the GUI, or at the command line:

Linux/Mac: ***./nC.sh -neuroml MyNeuroML.xml***

Windows: ***nC.bat -neuroml MyNeuroML.xml***

[14]http://www.hdfgroup.org/products/hdf5_tools). For more on HDF5 see section 4.3

3.5.5 NEURON

The NEURON simulation environment (Carnevale and Hines, 2006) is one of the most popular tools for creating detailed conductance based neuron and network models. The current version of NEURON natively supports import and export of cell morphologies in NeuroML Levels 1 and 2[15]. All releases of the application from version 6.0 onwards include these features, and updated files for this support can be retrieved from the NeuroML SourceForge repository. Details of the locations of the relevant files can be found at http://www.neuroml.org/neuron_tools.

The NeuroML export function in NEURON can be accessed when ModelView is open (Main NEURON Menu → Tools → ModelView). This functionality works best when just one morphologically detailed cell has been created from a cell template, as is the case with many single cell models on ModelDB. When exporting as a Level 2 file, the densities of channels and passive properties of the cell are included. To this end, the groups of sections with common electrical properties as generated by ModelView (ModelViewParamSubsets) are used as section groups, and a ***biophysics*** element is added to the exported NeuroML file.

The option for importing NeuroML morphologies is available via Main NEURON Menu → Build → Cell Builder → Management → Import → Import Button → NeuroML. Figure 3.11 illustrates a cell in MorphML which has been imported into NEURON.

ChannelML files can be converted to NMODL files using the latest XSL file for this mapping, e.g. **ChannelML_vXXX_NEURONmod.xsl**. This converts the XML into a mod file which can be compiled for use in NEU-

[15]This implementation was started by Michael Hines and I have helped kept it up to date as NeuroML has developed

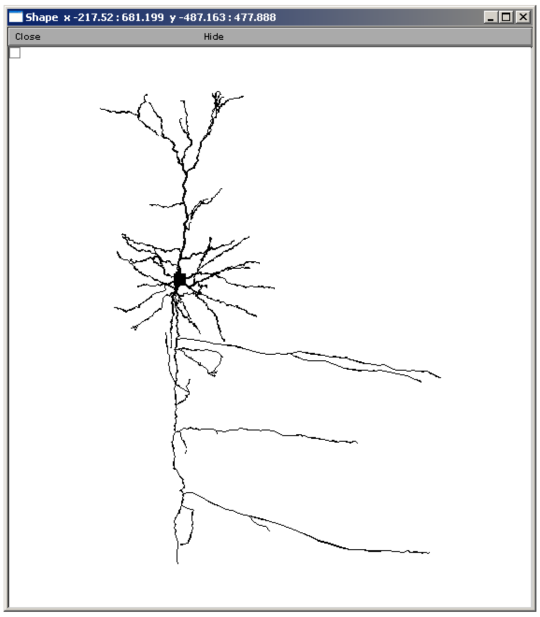


Figure 3.11: A pyramidal cell visualised in NEURON which was loaded in as a MorphML cell description (same cell as Figure 3.8).

RON. This conversion can be done with any XML tool for handling XSL file transformations. I have also developed a short script in Python to facilitate this[16]. This conversion is also possible via the NeuroML Validator web application (section 3.5.2, Figure 3.7). The flexibility of the NMODL language has meant that all channel and synaptic mechanisms covered by NeuroML to date are supported by NEURON. Level 1 cell morphologies (MorphML files) can also be converted to NEURON hoc files via that site with an XSL mapping (for example **MorphML_vXXX_NEURON.xsl**), but use of an interactive tool like neuroConstruct allows visualization and editing of the cells and export to NEURON and other formats.

3.5.6 GENESIS/MOOSE/GENESIS 3/Neurospaces

GENESIS (Bower and Beeman, 1997) is another popular platform for developing and simulating detailed cell and network models. It too has an active user community, documentation and a number of publications using

[16]http://www.neuroml.org/neuron_tools

models in this format. The most widely used version of the platform to date has been GENESIS 2. The Neurospaces (Cornelis and De Schutter, 2003) and MOOSE (Multiscale Object Oriented Simulation Environment, Ray and Bhalla (2008)) communities are actively developing new tools which will be compatible with scripts for this simulator.

GENESIS 2 does not natively support NeuroML. ChannelML files can be converted to GENESIS script files using the latest XSL file for this mapping, for example **ChannelML_vXXX_GENESIStab.xsl**. This conversion can be done with any XML tool for handling XSL file transformations, but a short script in Python as mentioned above for NEURON, can also be used for the transformation. This conversion is also possible via the NeuroML Validator web application (section 3.5.2, Figure 3.7).

The majority of (non kinetic scheme based) ion channels and non plastic synapse models supported by NeuroML can be mapped to GENESIS 2. There are a number of objects in GENESIS 2 for implementing plastic synapses (for example ***facsynchan***, ***hebbsynchan***), but these are based on different models of plasticity than the STP and STDP synaptic mechanisms used in NeuroML.

Level 1 cell morphologies (MorphML files) can also be converted to GENESIS script files via the validator website with an XSL mapping (for example **MorphML_vXXX_GENESIS.xsl**), but use of an interactive tool like neuroConstruct is also possible.

The MOOSE platform has extensive support for loading GENESIS 2 scripts. Some native support for loading NeuroML has been added by the developers of that platform, and the C++ library for this can potentially be reused by other applications. The latest details on MOOSE development are available at http://moose.sourceforge.net.

The Neurospaces project is developing major components of the GENESIS 3 platform. This will be a modular reimplementation of the core of GENESIS into a number of components for model loading and editing (including parsers for GENESIS 2 scripts and import functions for NeuroML cell models), a number of compartmental solvers, a scheduler for managing simulations, and command line and graphical interfaces. Some native support for passive cell models in NeuroML has been added. More details of the current developments towards GENESIS 3 can be found at http://www.genesis-sim.org and http://neurospaces.sourceforge.net.

3.5.7 PSICS

The recently developed neuronal simulator PSICS (Parallel Stochastic Ion Channel Simulator) allows simulation of detailed neuronal models which include stochastic ion channel transitions, and so can be used to examine the effect of low numbers of ion channels on neuronal firing behaviour (Cannon et al, 2010). This simulator has had an initial focus on single cell modelling, and while support for networks of cells is in development, NeuroML models incorporating synapses and networks cannot currently be run on this platform.

Level 1 morphologies can be imported natively by PSICS[17]. PSICS does not have an internal representation of cables, so MorphML ***cable*** elements (used for grouping segments) are only used to assign labels to points (taken from the ***segment*** elements), which can then be used for channel allocation. Figure 3.12 shows a cell morphology in PSICS.

PSICS natively reads a large subset of channel specifications in ChannelML 1.8.1 but does not support ligand gated channels (for example [Ca^{2+}]

[17]Native support for NeuroML morphologies and channels in PSICS was developed by Robert Cannon

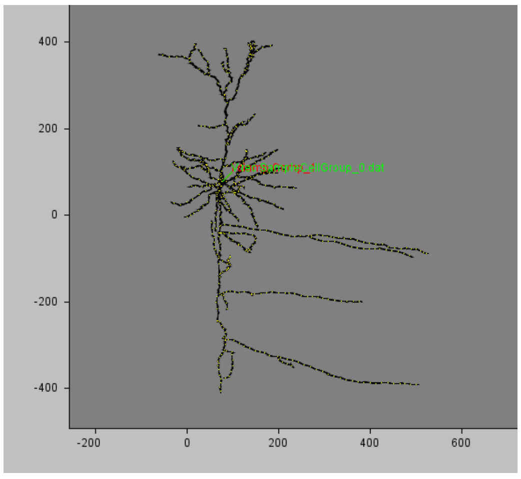


Figure 3.12: A pyramidal cell visualised in PSICS from a MorphML cell description (same cell as Figure 3.8). The figure is automatically generated by PSICS after running a simulation with the cell.

dependent K$^+$ channels), synapses or integrate-and-fire mechanisms. Supported ChannelML models can be converted to PSICS format using the latest XSL file for this mapping, for example **ChannelML_vXXX_PSICS.xsl**. This converts the XML into a PSICS compatible XML file and reports if the ChannelML file uses an unsupported construct. The conversion can be done with any XML tool for handling XSL file transformations, but a short script in Python to facilitate this is available[18]. This conversion is also possible via the NeuroML Validator web application (section 3.5.2, Figure 3.7).

3.5.8 PyNN

PyNN (Davison et al, 2008) is a Python package for simulator independent neural network development. This language allows simulators which have a Python based scripting interface (for example NEURON (Carnevale and Hines, 2006), NEST (Diesmann and Gewaltig, 2002) and Brian (Goodman and Brette, 2008)) to use the same set of scripting commands to create large scale neural networks. The PyNN approach to simulator independent model specification differs from that of NeuroML, as it is a language for

[18]http://www.neuroml.org/neuron_tools

the procedural description of model creation whereas a model specified in NeuroML is a declarative specification of the model structure. PyNN has also to date concentrated on descriptions of large scale networks of simplified neurons (type V in Figure 1.1B), whereas NeuroML has mainly been used for smaller networks of more biologically detailed cells. Thus, the two approaches are complementary.

PyNN scripts are intended to be used on multiple simulators with little or no modification. The one change that is usually needed is the line at the start of a PyNN script: ***from pyNN.neuron import **** should be replaced with ***from pyNN.nest import **** to use the simulator NEST instead of NEURON, etc. There is an initial implementation of a NeuroML module in PyNN by Andrew Davison which can be used to export the structure of the network created in the PyNN script, together with cell and synapse properties, to a NeuroML compliant file, as opposed to executing the network on the specified simulator.

A subset of models specified in NeuroML can be converted to valid PyNN scripts. Currently this is enabled by export of NeuroML based models from neuroConstruct which I developed. Due to the scope of models which can currently be expressed in PyNN only neuroConstruct projects containing the following can currently be mapped onto PyNN: single compartment cells containing only a passive conductance together with an Integrate and Fire based conductance; networks connected by alpha or single exponential waveform conductances (which could include STP or STDP based plasticity mechanisms); random spiking inputs.

Members of the PyNN and NeuroML development communities are involved in the INCF Program on Multiscale modelling, and updates of network representations (population layouts, connectivity schemes, etc.), as well as support for more generic representations of abstract cells in both PyNN and future versions of NeuroML will be coordinated through this forum (chapter 7).

3.5.9 NeuroMorpho.org

NeuroMorpho.org is a database of digitally reconstructed neurons (Ascoli et al, 2007). This resource can be used to retrieve reconstructed neuronal morphologies of multiple cell types from a number of species. The database can be browsed by neuron type, brain area, species, contributing lab, or cells can be searched for according to various morphometric criteria or the associated metadata.

There is a utility present on the site to view the cells in 3D (originally based on Robert Cannon's Cvapp[19]), which could be used to save the morphologies in NEURON or GENESIS format. I have recently updated this to allow these 6000+ reconstructed cells to be downloaded in NeuroML format.

3.5.10 TREES Toolbox

The TREES Toolbox[20] is an application developed in MATLAB which allows automatic reconstruction of neuronal branching from microscopy image stacks and generation of synthetic axonal and dendritic trees; visualisation, editing and analysis of neuronal morphologies; comparison of branching patterns between neurons; and investigation of how branching depends on local optimization of total wiring and conduction distance. This application was discussed in detail in a recent publication (Cuntz et al, 2010).

The latest version of the TREES toolbox includes functionality I have developed for exporting cells in NeuroML v1.x Level 1 (MorphML) or as a

[19]http://www.compneuro.org/CDROM/nmorph/cellArchive.html
[20]http://www.treestoolbox.org

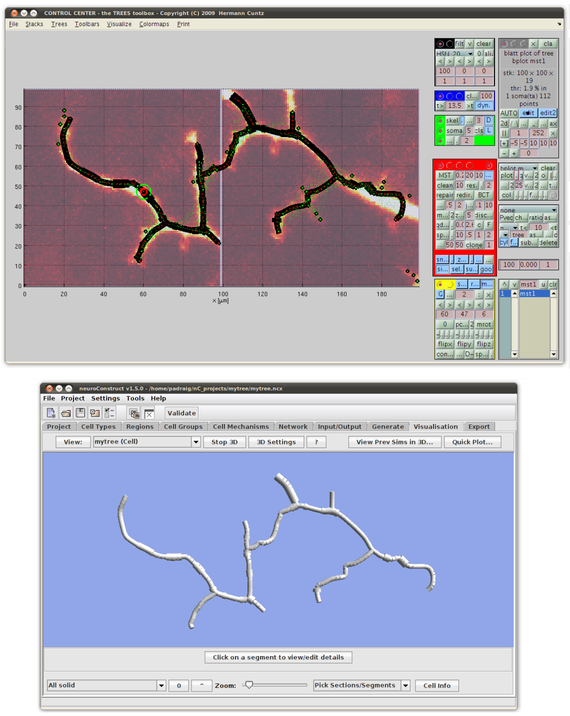

CONTROL CENTER - the TREES toolbox - Copyright (C) 2009 Hermann Cuntz
File Stacks Trees Toolbars Visualize Colormaps Print
x [μm]
neuroConstruct v1.5.0
File Project Settings Tools Help
Validate
Project Cell Types Regions Cell Groups Cell Mechanisms Network Input/Output Generate Visualisation Export
View: mytree (Cell)
Stop 3D
3D Settings
View Prev Sims in 3D...
Quick Plot...
Click on a segment to view/edit details
All solid
Zoom:
Pick Sections/Segments
Cell Info

Figure 3.13: The top screenshot is of the TREES toolbox control center, showing a neuron reconstructed from a set of image stacks. The reconstruction can be automatically generated based on a number of criteria set through the interface, and the identified points can be manually edited if needed. The lower screenshot shows the morphology loaded into neuroConstruct after being exported from the TREES toolbox in NeuroML.

NeuroML v2.0 morphology file (see Figure 3.13).

3.5.11 CX3D

CX3D[21] can be used for simulating the growth of neurons in 3D, both stochastically and in response to the presence of neurochemical attractors (Zubler and Douglas, 2009). The developers of this application initially included an option for export of soma positions in NetworkML, and I updated this to allow export of the full morphologies of the grown cells and the network structure in a NeuroML Level 3 file (see Figure 3.14)[22].

[21]http://www.ini.uzh.ch/~amw/seco/cx3d/

[22]An updated version of the CX3D application with greater NeuroML support is available here: http://sourceforge.net/apps/trac/neuroml/browser/CX3D2NeuroML

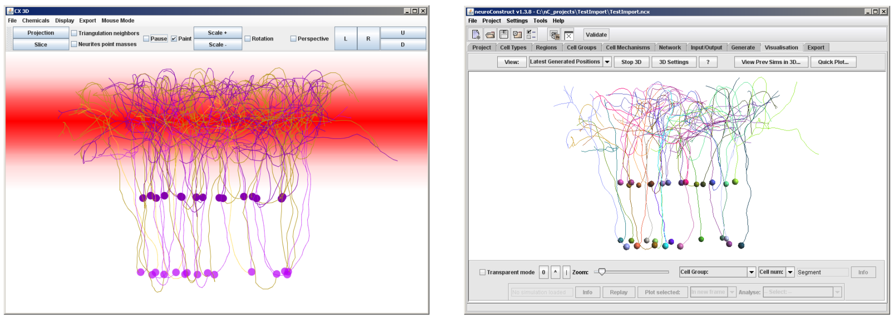


Figure 3.14: A screenshot of a network generated in CX3D (left). The axons of both the upper and lower cell groups are attracted to the chemical substance whose concentration is shown in red. The network was exported in NeuroML format and loaded into neuroConstruct (right).

3.5.12 NETMORPH

NETMORPH (Koene et al, 2009) is an application for the developmental generation of 3D large-scale neuronal networks with realistic neuron morphologies. The growth cones of individual neurites are simulated and can elongate, branch and turn to produce realistic neuronal morphologies. Synaptic networks can also be generated from these generated neurons.

I have developed an initial converter from the text file output of NETMORPH into NeuroML[23], which allows these generated networks to be used in other NeuroML compliant applications.

3.6 NeuroML version 2.0

The current version of NeuroML can be used to specify a wide range of neuronal models ranging from single cells to complex networks in 3D containing multiple cell types. Models have been converted to this format of cells and microcircuits from multiple brain regions (chapter 5). A number of ways

[23]http://sourceforge.net/apps/trac/neuroml/browser/NETMORPH2NeuroML

were identified however, in which the language could be further developed to increase the flexibility for developing new types of models and this has led to NeuroML version 2.0.

3.6.1 Limitations of NeuroML v1.x

NeuroML v1.x focussed on conductance based models of mainly multicompartmental neurons, due in large part to the target modelling environments NEURON and GENESIS, and the types of modelling carried out by the main parties involved in its development. Less well supported were the abstract neuron models (e.g. various types of Integrate and Fire neurons) favoured by large scale simulators like NEST. Also, the dynamical behaviour of NeuroML v1.x entities (e.g. channel models, synapse types) was defined in the text of the specification (Supporting Text S1 of Gleeson et al (2010a)) and developers of parsing applications needed to know these definitions to interpret the NeuroML model descriptions correctly. One of the key limitations of v1.x was the difficulty in developing new type of models without requiring an update to the core language.

While some basic network templates were present in v1.x (Figure 3.6), this did not cover the range of network descriptions (including hierarchical descriptions of columns, hypercolumns etc.) often used in papers to describe model network structure. The instance based description of networks in v1.x was useful for exchanging information on generated 3D networks, but more work was required on compact descriptions of generic network structures.

The Systems Biology community have been actively developing tools and databases of models using standardised languages (SBML and CellML), which while having a different focus to NeuroML (e.g. metabolic pathways and subcellular signalling networks), were of interest to researchers creating multiscale cell models. NeuroML v1.x did not offer any support for integrating such models into neuronal simulations.

3.6.2 New features of NeuroML v2.0

Key changes over v1.x which are being developed for v2.0 include:

- A mechanism for defining extensible ComponentClasses, which provide unambiguous, machine readable descriptions of the behaviour of model Components such as ion channels, synapses, abstract cells and other physical entities. This low level model description language provides the flexibility to build a wide range of biophysical models.
- Import and export functions for models in SBML and CellML, including detailed cellular signalling pathways.
- Greater support for templates of network structure, in line with work in the INCF Multi-scale Modelling Program.
- A library for reading/writing NeuroML files (libNeuroML) and exporting to/importing from multiple languages.
- Links to structured annotation formats (for example Systems Biology Ontology[24], Gene Ontology[25], NeuroLex, etc.).

NeuroML v2.0 is in active development but significant parts of these new features have already been implemented.

3.6.3 Components and ComponentClasses

NeuroML v2.0 is designed to have machine readable definitions of the core model components, to facilitate unambiguous interpretations of the model

[24] http://www.ebi.ac.uk/sbo
[25] http://www.geneontology.org

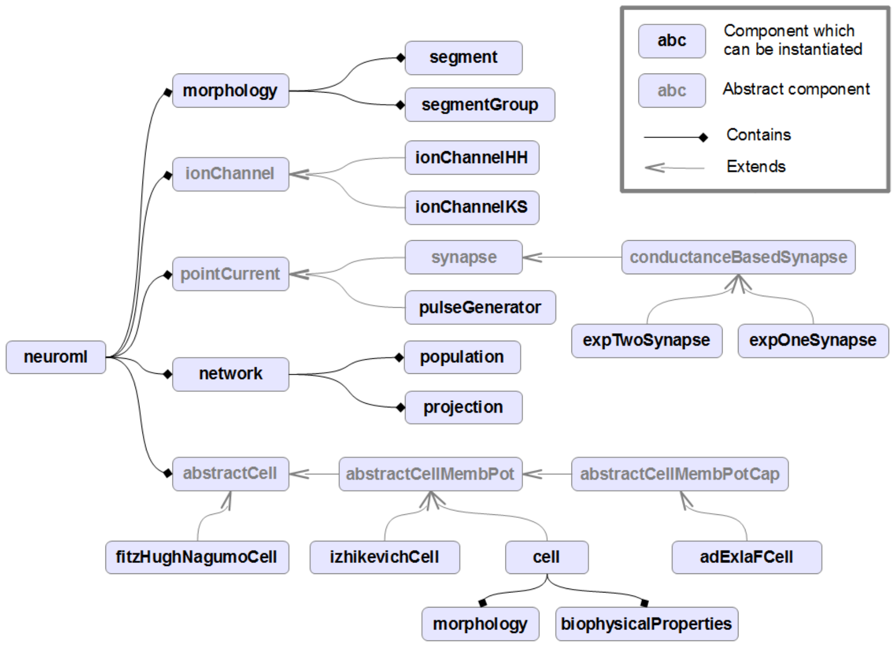


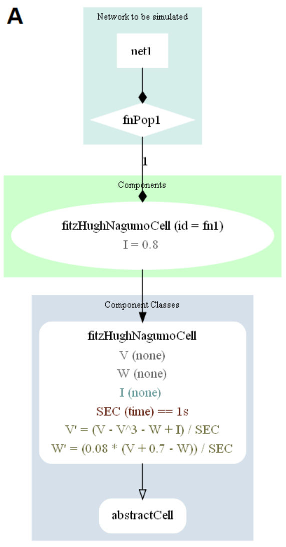
Figure 3.15: Hierarchical structure of NeuroML v2.0. Names of Components which can be instantiated (black text) correspond to the XML element names used in valid NeuroML v2.0 files.

behaviour across implementations. The overall structure of v2.0 is roughly the same as v1.x (Figure 3.15) with a top level ***neuroml*** element, containing channel, synapse and cell definitions and networks of populations. The key difference is that a dynamical model component has its behaviour described explicitly in a ComponentClass[26] (instantiations of which are referred to as Components).

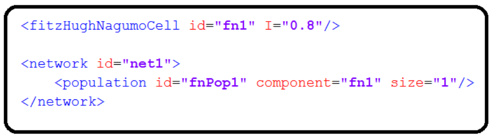
The specification language (also in XML) for defining ComponentClasses in NeuroML v2.0 is LEMS, the Low Entropy Model Specification language[27]. Composition of and extensibility of ComponentClasses is at the core of

[26]The term ComponentType was originally used for this concept.
[27]Robert Cannon initially developed the LEMS language and interpreter and I updated NeuroML v2.0 to use the language for defining its core model components.

A



B



```
<fitzHughNagumoCell id="fn1" I="0.8"/>

<network id="net1">
    <population id="fnPop1" component="fn1" size="1"/>
</network>
```

C

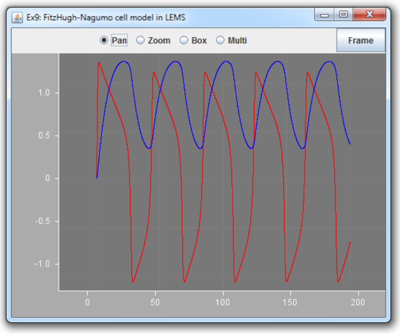


Figure 3.16: FitzHugh-Nagumo model cell specified in NeuroML v2.0. A) A network specifies that it contains a single ***population*** containing an instance of a ***fitzHughNagumoCell***. The definition for the behaviour of this Component is contained in a ComponentClass. This graph has been automatically generated from the XML definition of the ComponentClass (not shown) and the network definition creating the Component instance, shown in B. B) The XML corresponding to the Component and network in A. C) The model after being executed by the LEMS interpreter, showing behaviour of the state variables V (red) and W (blue).

LEMS. Cells, channels, etc. can have children (e.g. ion channels can contain multiple gates) and can extend base types, inheriting their exposed variables (e.g. v, membrane potential with units voltage). In this way the language can be easily extended, with simulators knowing that any new ComponentClass extending ***synapse*** exposes a current and receives spike events, etc. Figure 3.15 shows the current hierarchical structure of v2.0.

Figure 3.16 illustrates an example of how explicit definitions of a cell model is specified in v2.0. The behaviour of the model component (in this case a simple FitzHugh-Nagumo cell model (FitzHugh, 1961)) is specified in a ComponentClass (A, bottom), with the state variables (V, W) specified along with their dynamics in time in terms of the fixed parameters of the model. An instance of this, a Component, is created in a network by setting the free parameter (I, injection current).

The definition of the ***fitzHughNagumoCell*** ComponentClass is included in the core of NeuroML v2.0 and the XML used to create a simple network of one cell is shown in Figure 3.16B. Other abstract neuron models such as Integrate and Fire (I&F) neurons, and two state variable extensions of this such as the Izhikevich neuron model (Izhikevich, 2003), and Adaptive Exponent I&F model (Brette and Gerstner, 2005) are also currently supported (Figure 3.15).

An initial implementation of a parser for LEMS based model definitions has been built in Java (the LEMS Interpreter) which can also execute models with a simple numerical integration routine. Figure 3.16C shows the LEMS Interpreter after executing the FitzHugh-Nagumo model.

3.6.4 libNeuroML

A library, libNeuroML, has been developed in Java which supports LEMS at its core. Coupled with the core set of NeuroML ComponentClass definitions it can read and write NeuroML v2.0 models, or any LEMS model (Figure 3.17). Import and export functions have been created to read/write other model description formats (e.g. NineML or SBML, an example of which is given in section 4.6), generate scripts for simulators (e.g. NEURON) or them convert to other model description or graphical formats. libNeuroML has been integrated into neuroConstruct to facilitate simulation management, visualisation and analysis of v2.0 models.

More details on the future directions of the NeuroML initiative are discussed in chapter 7.

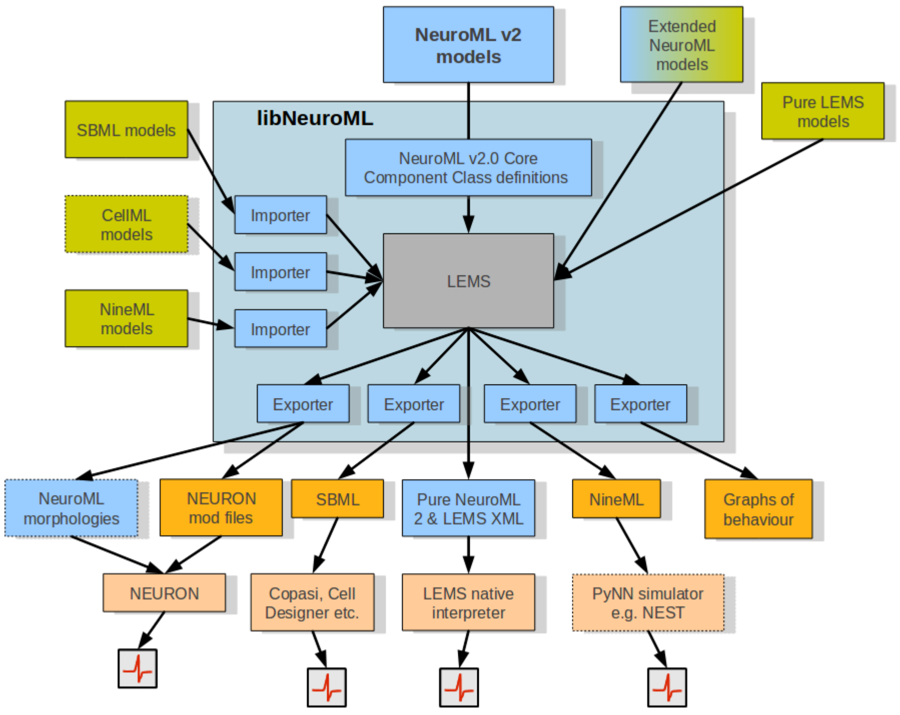


Figure 3.17: Import and export formats in libNeuroML. The dotted boxes represent planned functionality, but all other interactions are available in prototype form.

3.7 Conclusions

The NeuroML initiative has developed much since I became involved in 2004. Concentrating on creating a specification for a language which worked with existing simulators and published models has lead with NeuroML v1.x to model component portability and cross simulator validation of models which had not previously been possible, and which will be essential for greater sharing and reuse of models between researches. Close integration of the language with neuroConstruct has meant that users can get the interoperability and accessibility benefits of the language without having to directly handle XML files. Making the specifications open has led to a number of application developers independently implementing support for the language.

The next version of NeuroML builds on the pragmatic solutions developed for v1.x and will allow cellular neuroscientists interested in detailed conductance based models, theoreticians developing abstract network models and systems biologists interested in subcellular signalling to use a common language and set of tools for exchanging models and communicating ideas.

Chapter 4

Simulation Management for Large Scale Network Models

This chapter presents some of the issues involved in creating and managing large scale models of neuronal networks and outlines the tools and methods developed as part of my research to facilitate the generation of scripts for such simulations and the management of simulation data.

I will first present an overview of the motivating factors for developing large scale network models, the existing solutions for creating these and some of the issues related to generation and management of these simulations. I will then discuss the extension to neuroConstruct for generating scripts for Parallel NEURON, solutions for storing network structure and simulation results in compact binary formats, the Python interface for controlling neuroConstruct through scripts, and a new way to incorporate subcellular signalling descriptions in SBML into neuronal models.

These tools and methods have been applied to the development of highly detailed cerebellar and thalamocortical network models as described in Chapter 5, and have all been made freely available to other members of the neuroscience community for use in their research.

4.1 Issues related to large scale network simulation

One question which should be addressed at the outset is: why model neuronal networks on a large scale[1]? A common criticism is that if a single cell model with multiple conductances can have on the order of hundreds of free parameters, then a network of thousands of cells can have so many that is would be possible to tune the network to produce any desired behaviour.

One of the main reason for incorporating a high level of biophysical detail into models is to investigate the physical plausibility of a proposed explanation for a phenomenon. If a conceptual model of how a network behaves involves multiple weak synaptic inputs into each cell, then this needs to be incorporated into a computational model. Adding many anatomical and physiological properties allows questions to be asked about which are essential for reproducing the experimentally observed behaviours in the model system. Large scale models can also be useful for giving insight into very complex interactions in a more controllable system.

When cells are synaptically connected, larger networks will better reflect the physiological inputs received by a single cell. While a cell's response is normally the result of thousands of weak synaptic inputs, in many models the number of inputs are scaled down and the strength of each scaled up. Also, the sparseness of connectivity as measured in real circuits may be sacrificed if insufficient numbers of cells are present, overemphasising pairs of

[1]The concept of "large scale" also needs to be defined. Lansner and Diesmann (in press) point out that this concept changes with developments in computer technology, and could be defined as whatever size network begins to task a standard workstation (and so the types of network classed as large scale will change over time). A better definition could be that the size of individual distinguishable units is much smaller than the system as a whole: whereas a "whole brain" cognitive processing model with 20 interacting areas would not be considered "large scale", a single cortical column model with 10,000 cells would be.

cells' influence on one another. In networks with too few cells, artificial background noise is often added to provide a "realistic" *in vivo* like environment, whereas in larger networks a cell's inputs are more likely to be representative of simulated cells which are responding to the same overall network activity. Larger scale networks can be used also to investigate the generation of macroscale phenomena, such as Local Field Potential (LFP) activity (Diwakar et al, 2011) or voltage sensitive dye imaging signal (Chemla and Chavane, 2010).

The usefulness of using large populations of cell models can also be addressed by considering a detailed cell model with a given set of conductances, created using experimental data from a number of single cell recordings. If there is no stochastic elements to the model (as is normally the case) the cell will only ever produce a specific output for a given input: no variability will be displayed by the cell despite the fact that the data used to produce the model had this. A population of cells on the other hand, with parameters taken from distributions so that the average spiking rate reflected the mean recorded in the data, could be considered a better encapsulation of its biological counterpart (Marder and Taylor, 2011).

These points suggest that there are definite advantages in creating and studying large scale network models. Care does indeed have to be taken however, and the modeller needs to be explicit about the information used in creating the network. A set of synaptic weights with a given mean and standard deviation comprises just 2 free parameters whether the network has 100 or 10^9 connections of that type. As long as the whole network can be described in such a way, and presented in a concise format, then it is open to critical review by others, parameters can be adjusted and the effect on overall behaviour studied. This will lead to a more scientific assessment of the worth and accuracy of such models, and the ability to build on then and use them for asking new questions about neuronal function.

Critical to this process though is the infrastructure to generate these large scale models from the compact descriptions, to simulate them, to handle the large amounts of data produced, to visualise the results and to analyse their behaviour.

4.1.1 Simulations on multiple processors

A key element in the creation and analysis of large scale network models is the simulation platform used. These range from parallel versions of widely used open source simulators, “home grown” solutions which are developed over time by small sets of researchers, to less open systems which make claims to model massive, realistic networks, the specific details of which are not always made publicly available.

Some of the initiatives to enable large scale neuronal simulations include:

Parallel NEURON: NEURON continues to be the most popular platform for creating detailed, conductance based neuronal models. One of the most significant recent developments in NEURON is the ability to run simulations across multiple processors (Hines and Carnevale, 2008; Migliore et al, 2006). This parallel functionality allows near linear speedup of network simulations, i.e. a large network spread across 100 processing cores will run approximately 100 times as fast as the same network on one processor. Recently there has also been the option to split cells with large numbers of sections efficiently across multiple processors (Hines et al, 2008a). This platform is the main target of large scale simulations in neuroConstruct and is discussed in more detail in section 4.2.1.

PGENESIS: The GENESIS simulation environment (Bower and Beeman, 1997) also has had a version designed for parallel simulations. The core of GENESIS was extended to allow networks of cells to be distributed across multiple compute nodes, with the underlying communication handled by PVM (Parallel Virtual Machine) or MPI (Message Passing Interface). PGENESIS was developed at the Pittsburgh Supercomputing Center[2]. The two initiatives which are part of developments towards GENESIS 3, Neurospaces (Cornelis and De Schutter, 2003) and MOOSE (Multiscale Object Oriented Simulation Environment, (Ray and Bhalla, 2008)) are both developing parallel simulation support.

NEST: The NEST (NEural Simulation Tool) simulator (Diesmann and Gewaltig, 2002) is an application for simulating networks of neurons of biologically realistic size (on the order of 10^6 cells, but much larger network simulations are planned) and is being developed as part of the NEST Initiative[3]. The emphasis is on efficiency of storage of the network information to allow the investigation of the dynamics of large scale networks. Neuronal elements in these networks (usually single compartment Integrate and Fire neurons) can be connected with a range of synaptic models including ones for short term plasticity (STP) and spike-timing dependent plasticity (STDP). To date the NEST Simulator has been mainly used for investigations of network activity in generic cortical structures (Kumar et al, 2008), and for studying the technologies needed for large scale parallel network simulations (Morrison et al, 2005; Plesser et al, 2007). NEST, like NEURON, is supported by the PyNN scripting language (Davison et al, 2008) for specifying networks in a simulator independent manner (section 3.5.8).

Blue Brain Project: This ambitious project aims to create a realistic model of the cortical column (Markram, 2006) to run on a Blue Gene/P

[2]http://www.psc.edu/Packages/PGENESIS
[3]http://www.nest-initiative.org

supercomputer. The project is incorporating large amounts of neuronal reconstructions and electrophysiological data from the Markram lab and elsewhere to create the model. While much of the model details are not yet public, the main simulation platform being used, Parallel NEURON, and the new algorithms for distributed computing developed for it, are freely available.

Home grown simulators: A number of simulation platforms have been developed by independent research labs in the past few years for specific modelling projects. The SPLIT simulator, developed in KTH, Sweden, has been used for large scale network simulations of associative memory in Layer 2/3 of the cortex (Djurfeldt et al, 2008). While this has been useful as testbed for developing and researching these large scale models, the simulator has not been widely used outside of the group who developed it. Roger Traub has developed FORTRAN based simulations of thalamocortical network models (Traub et al, 2005) which are distributed across multiple compute nodes. While the original code is freely available, not many other researchers have modified or reused this model code directly. The large scale thalamocortical model of Izhikevich and Edelman (2008) contains reduced models of multiple cell types across all cortical layers and realistic connectivity and exhibited alpha and gamma rhythms and cell groups with up and down states.

Hardware based simulations: While most of the above simulation platforms have used standard cluster hardware, there have also been various initiatives to create large scale simulations using hardware based solutions. These include development of “neuromorphic” hardware, where VLSI (Very Large Scale Integration) systems are designed to mimic the signal processing in biological neuronal networks. Initiatives in this field include: the EU

BrainScales project[4] (and its predecessor FACETS) which is using PyNN to specify the neuronal network properties, the SyNAPSE project[5], funded by DARPA in the US; and NeuroGrid project of Kwabena Boahen's lab in Stanford[6].

GPU based solutions: There are an increasing number of projects looking at utilising the large number of processing cores on the Graphics Processing Units (GPU) on most modern video cards for parallel network simulations. Dedicated development environments including NVIDIA's CUDA are facilitating development of applications on such hardware. The Brian simulator developers have started code generation for GPUs (Goodman, 2010), and other dedicated simulators are in development, e.g. NeMo[7] (Fidjeland and Shanahan, 2010) and GeNN[8].

As this overview shows, there are a number of technical approaches being taken to facilitate the simulation of large scale neuronal networks. Due to the nature of the goal ("reverse engineering the brain"), there has been a lot of public coverage for some initiatives which belies the fact that most of the projects involve software which is at most prototype research code. All of these large scale modelling initiatives can learn from one another and it will require open discussion and testing of one another's code to create a solid foundation for biological scale neuronal simulations.

[4]http://brainscales.kip.uni-heidelberg.de
[5]http://www.neurdon.com/about-synapse
[6]http://www.stanford.edu/group/brainsinsilicon
[7]http://www.doc.ic.ac.uk/∼akf/nemo
[8]http://sourceforge.net/projects/genn

4.1.2 Large data set management

A key issue related to the use of large scale simulations in research is management of the large data sets used/generated by the simulations. In a large scale network of even a few thousand cells the number of synaptic connections present in the network can number in the millions. Pre- and postsynaptic cell IDs along with the synaptic weight will need to be stored. For larger networks, the list of cell positions can also occupy a significant amount of memory. For many simulators the solution is to store only the algorithms used to generate the networks, and the seeds used to generate specific simulation instances. The exact network structure/state will need to be recorded however, if the network is to be analysed/visualised after the simulation completes.

With increasing numbers of cells in models comes an increase in the potential size of the data which can be produced by the simulations. While sampling of the data can be used to compress the output of the simulations (e.g. only record spike times), an increase in the detail of individual cells means that there are potentially more interesting variables to record. In contrast to networks of Integrate and Fire neurons, a researcher using a highly detailed simulation such as the thalamocortical column simulation (section 5.2.4) can be interested in each neuron's subthreshold activity (to investigate up/down states), activity in apical dendrites in all pyramidal cells, intracellular calcium concentrations along cell dendrites or time evolution of synaptic weights in the millions of synapses present.

These large data sets will need to be efficiently managed, especially if simulation and analysis are taking place on different networked machines.

4.1.3 Limitations of GUI based approach

While a graphical user interface is very convenient for a modeller to investigate and visualise the structure of a network and can be useful for other researchers to quickly familiarise themselves with published models, there are practical limitations to the types of questions that can be asked using just the GUI. Systematic parameter searching of ill constrained parameters may require a large number of simulations for all of the permutations involved.

While it is possible to add features in a GUI to automate this process (e.g. selecting the conductance densities to alter, load a the target data file, drop down menu of optimisation algorithms, etc.) it is impossible to predict every option required by a user and to add support for them graphically. These user requirements point to the need for a greater level of access to the model parameters "behind" the GUI from within the scripts used for model optimisation.

4.1.4 Multiscale modelling

Changes at one biological level (e.g. alterations in plasticity behaviour of a synapse) can lead to effects at a higher level (e.g. overall network synchrony). Computational models can be used to investigate the functional underpinnings of these effects. This type of multiscale modelling will increasingly rely on use of model components from other fields when a researcher's own expertise lies at a different level. There is also the issue that different modelling languages and techniques are used by different communities. SBML (Hucka et al, 2003) and CellML (Lloyd et al, 2004) have been used for many years to create models of biological systems from sub- and intercellular signal transduction, cell metabolism, gene regulation and even whole cell models.

Both communities are at pains to point out that the languages can be used for more generic dynamical systems, and both have repositories of models built around their own formats (BioModels database (Li et al, 2010) and CellML Model Repository (Lloyd et al, 2008) respectively). Both of these allow export of their models in other formats (e.g. MATLAB). A key issue when building multiscale models will be to ensure model components from different communities can be integrated and reused as transparently as possible.

4.2 Support for parallel network simulations in neuroConstruct

neuroConstruct was originally developed to generate simulation scripts for the NEURON simulator. In time support for GENESIS and other simulators was added as the features in neuroConstruct for handling cells, channels, synapse models and network structure could be readily mapped to the equivalent entities in these simulators (section 2.3.1).

When initially investigating options for generating simulations of conductance based neuron models on parallel computing platforms, the NEURON community was most actively involved in this area, and was chosen as the initial target platform for parallel code generation from neuroConstruct. Functionality for this was added to the application so that the features required for any parallel simulator (assignment of cells to compute nodes, submitting scripts to/retrieving data from remote clusters) were not NEURON specific, which is designed to ease addition of support other parallel simulators (e.g. MOOSE, NEST) to the application at a later stage.

4.2.1 Parallel NEURON

The NEURON simulator has had support for parallel network simulations based on MPI for a number of years (Hines and Carnevale, 2008; Migliore et al, 2006). This functionality was added to the simulator "under the hood", so that the user can reuse as much of their scripts for serial execution as possible. The user only has to decide on the number of compute nodes, assign cells (with a unique global identifier, ***gid***) to the nodes, create connections between presynaptic locations and postsynaptic response objects, and run the simulations. New classes in NEURON (in particular ParallelNetManager) facilitate setting up these connections between objects on different nodes, and NEURON internally handles passing of spike events between distributed sources and target locations during the simulation. Full details of the recommended use of the parallel functionality in NEURON have been published (Hines and Carnevale, 2008).

The key to getting speedup of simulations across multiple compute nodes is finding the minimum value of axonal and synaptic transmission delay (e.g. 5ms) of all synaptic connections, and allowing cells on all nodes to execute in parallel for this time. After this time, the simulation is paused and all new spiking events are passed using the standard ***MPI_Allgather*** method. The simulation restarts and the spike events are inserted at the appropriate point.

The developers of this functionality admitted they were initially surprised as how easy it was to get better than linear speedup of simulations (Migliore et al, 2006) from a simulator designed for serial execution. The time spent distributing the spikes to all nodes is normally much less than the compute time between ***MPI_Allgather*** calls. Superlinear speedup of simulations (i.e. a greater than n times increase when the number of nodes is increased by n) is possible in many cases due to a smaller fraction of a network of a given size being present on each node, and so making better use of the faster cache memory (Morrison et al, 2005).

The large user base of NEURON has ensured that this feature of the simulator is well tested and used already in a number of published models (Solinas et al, 2010; Vervaeke et al, 2010). Development of parallel network functionality has been supported in part by the Blue Brain Project (Markram, 2006). This collaboration has also resulted in functionality for splitting cells with complex morphologies between compute nodes for better load balancing (Hines et al, 2008b,c).

4.2.2 Generic parallel simulation handling in neuroConstruct

Support for generating parallel network simulations has been incorporated into the standard neuroConstruct release. In a neuroConstruct project, cell groups are defined consisting of a cell type, the 3D Region in which they are placed and a specification of the packing pattern to use (Figure 4.1A, B). Simulation configurations consist of sets of cell groups, together with network connections, electrical inputs and lists of variables to plot and/or save during the simulation (Figure 4.1C). The various configurations of the parallel computing environment in which network simulations can be run are separate from any specific simulation configuration (Figure 4.1D). The two key terms used to define parallel configurations are: **host**: a single machine with its own hostname, a number of processors and a single shared memory; and **processor**: an individual CPU on a host. The various types of parallel configuration can include: a single processor on the local machine, a number of local processors, or a set of hosts each of which can have multiple processors. There is an assumption that all processors have equal computing

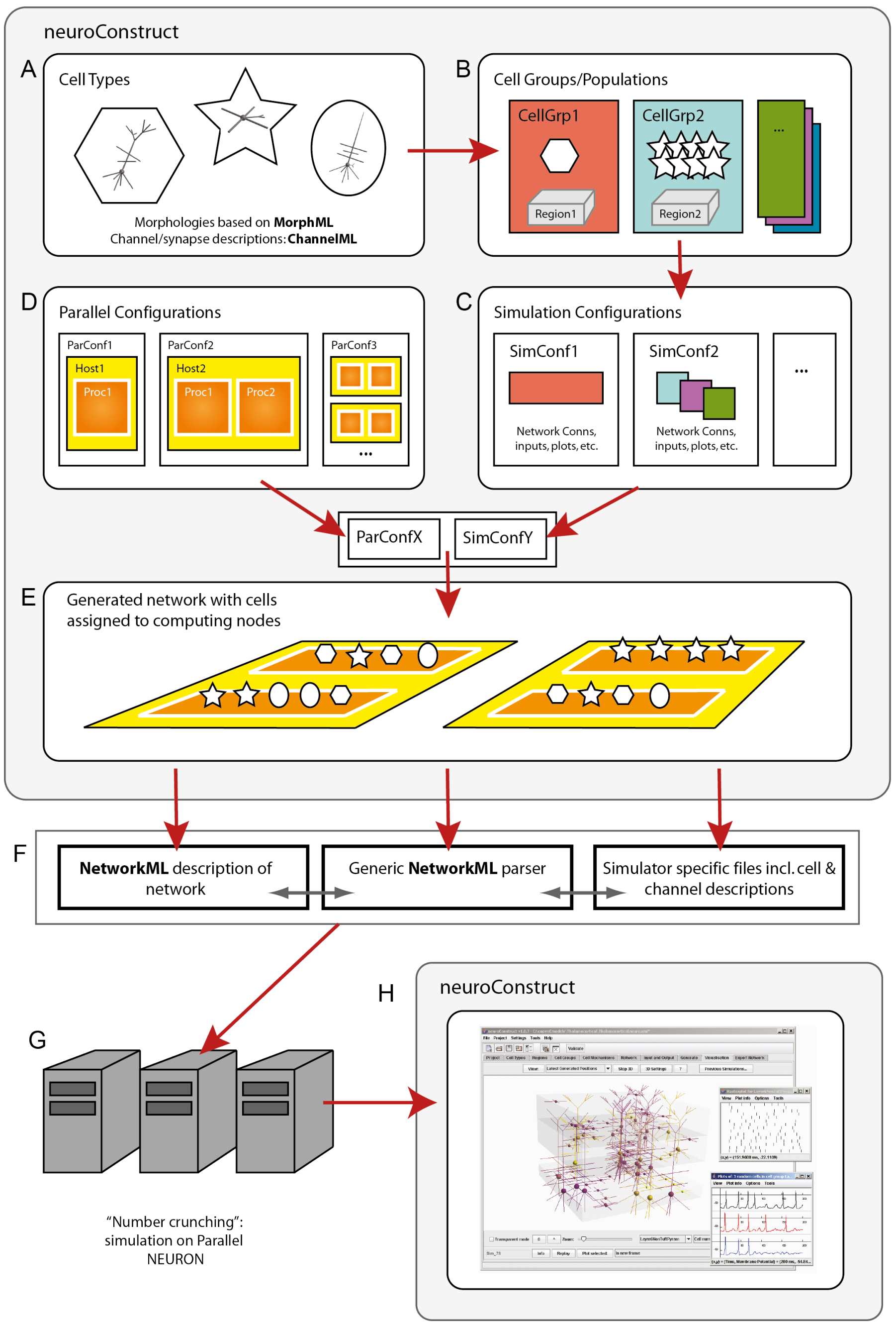

neuroConstruct
A
Cell Types
Morphologies based on MorphML
Channel/synapse descriptions: ChannelML
B
Cell Groups/Populations
CellGrp1
Region1
CellGrp2
Region2
...
D
Parallel Configurations
ParConf1
Host1
Proc1
ParConf2
Host2
Proc1
Proc2
ParConf3
...
C
Simulation Configurations
SimConf1
Network Conns, inputs, plots, etc.
SimConf2
Network Conns, inputs, plots, etc.
...
ParConfX
SimConfY
E
Generated network with cells assigned to computing nodes
F
NetworkML description of network
Generic NetworkML parser
Simulator specific files incl. cell & channel descriptions
G
"Number crunching": simulation on Parallel NEURON
H
neuroConstruct

Figure 4.1: Schematic illustrating the independent specification of network structure in simulation configurations and the target distributed computing environment in parallel configurations. A NetworkML representation of this network structure is then generated along with simulator specific code. See text for more details.

power, but hosts can have varying numbers of processors. An ID starting at 0 is assigned to each of the processors (***node_id***).

At the time of generation, the cells and connections for a particular simulation configuration are created first, and only afterwards are the cells assigned to the processors as defined in the chosen parallel configuration. At present the assignment of cells to compute nodes takes place in a round robin fashion, i.e. cells in a group sequentially get assigned an increasing processor id, which returns to zero after all processors get a cell, and the distribution continues. This ensures an even distribution of cells on each processor if the number of cells is much larger than the number of processors, as is generally the case. The cell/processor assignments within neuroConstruct are independent of the mapping to a specific simulator (Figure 4.1E).

4.2.3 Parallel NEURON generation from neuroConstruct

There are two main options for generating the NEURON scripts for the parallel simulation: generating hoc files (NEURON's native scripting format) with all cell placement and network connections written explicitly in these; or saving the network structure in NetworkML (either in an XML or HDF5 file), and parsing this file to create the network. While the hoc option is "pure" NEURON code and does not require the version of NEURON compiled with Python, it does produce very large simulation scripts. The XML/HDF5 option keeps all of the network structure in one file in a compact format.

In the latter case, the NetworkML file specifying the network structure (incorporating details of how the cells should be distributed across the nodes) is created (Figure 4.1F, left), along with a generic NetworkML parser written in Python (appropriate for any NetworkML file and any simulator supporting Python, Figure 4.1F, middle) and files specific for the simulator, to create the cells and manage the simulation (Figure 4.1F, right). These scripts are then executed either on the local machine or on a remote computing platform (see below) and simulation results can be reloaded back into neuroConstruct for visualisation and analysis in the same way as serially executed simulations.

4.2.4 Interaction with remote computing hardware

It is also possible to generate parallel simulations for execution on remote machines which don't share a file system. Access to the machine without a password using ***ssh*** is required[9]. Figure 4.2 shows the set of interactions for carrying this out. The scripts generated by neuroConstruct are zipped (in a gzipped tar file), transferred to the remote machine by ***scp***, unzipped, NMODL files compiled and the simulation started. An optional extra step is the generation of a submission script for a batch queueing system which will schedule the simulation for execution on distributed computing resource. Commonly used batch submission systems currently supported and tested on UCL's High Performance Computing resources (Table 4.1) include TORQUE[10] (as used on the UCL Legion cluster) and Sun/Oracle Grid Engine[11] (as used on the Silver Lab's private cluster). The neuroConstruct user can then check which simulations have completed through the GUI and pull them back for local analysis.

These features of neuroConstruct were used by Koen Vervaeke in the Silver Lab for executing simulations of an electrically coupled Golgi cell network (Vervaeke et al (2010); section 5.1.4) on UCL's Legion supercomputer.

[9]Directions for this can be found with the instructions for installing Parallel NEURON: http://www.neuron.yale.edu/neuron/static/docs/help/neuron/neuron/classes/parcon.html

[10]http://www.clusterresources.com/pages/products/torque-resource-manager.php

[11]http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html

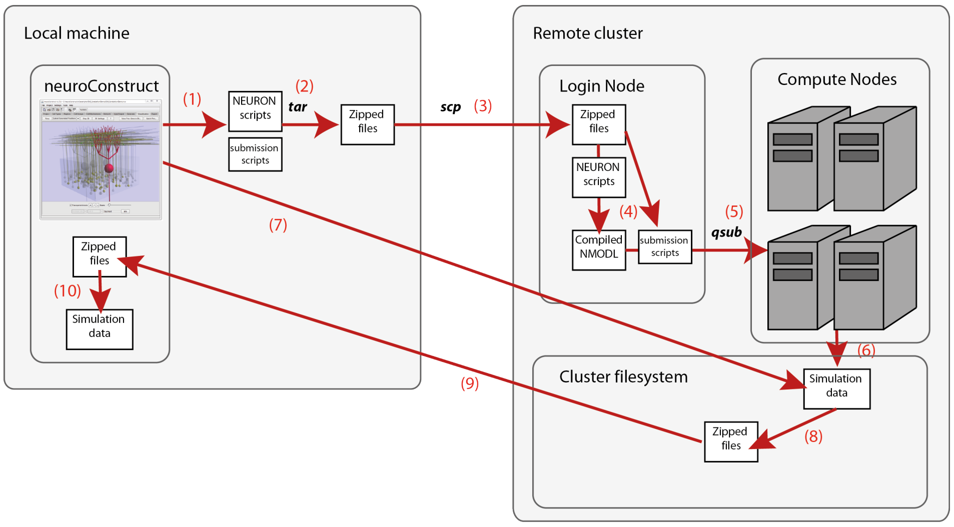


Figure 4.2: Remote execution of parallel scripts generated by neuroConstruct. Executable scripts for a parallel network simulations are generated on one machine, and neuroConstruct performs the required steps to move the scripts to a remote machine and submit them for execution to a cluster. 1) Files are generated based on the chosen simulation configuration/parallel configuration combination. The NEURON scripts are generated, as well as scripts containing the commands to submit the job to the chosen computing resource. 2) These files are zipped locally and 3) are transferred to the login node of the remote system. 4) They are unzipped, and the NEURON mod files are compiled for the local hardware architecture. 5) The job is submitted to the queue for execution on the compute nodes (using ***qsub***). 6) The simulation finishes and the data is stored on the remote storage system. 7) When the neuroConstruct user wants to retrieve the completed simulation, the application checks in the remote location where the simulation files should have been generated. 8) If the simulation has been successful, the data is zipped again and 9) copied back to the local machine, 10) where the data can be unzipped and is accessible for visualisation/analysis in the same way as locally run simulations.

Legion HPC cluster	2560 cores each with 4GB RAM per core Organized into 10 Computational Units of 256 cores InfiniPath chip-to-chip connectivity Shared resource across all UCL departments
Silver Lab cluster	240 compute cores 20 x 8 core (2.27 GHz, 24GB RAM) nodes 10 x 8 core (2.93 GHz, 48GB RAM) nodes Infiniband connectivity Dedicated Silver Lab resource

Table 4.1: UCL based computing resources

This was all carried out through the GUI and minimal experience with distributed computing or knowledge of scripting for Parallel NEURON was required. The functionality for automated generation of parallel simulations by neuroConstruct has also been successfully tested by a number of other labs around the world (section 7.3.1).

4.2.5 Performance measures

To test the generation of parallel code for NEURON, network simulations were generated from 2 models which had been converted for use in neuroConstruct, a 3D granule cell layer model and a thalamocortical network model. The granule cell layer model is based on an extension to 3D of the 1D granule cell layer model of Maex and Schutter (1998). More details of this model are given in section 5.1.4. It consists of randomly spiking mossy fibre inputs driving granule cells with parallel fibres, and Golgi cells which form an inhibitory feedback loop with the granule cells. A Python script for opening the neuroConstruct project containing the model was created, which could adjust the size of the network, generate it and send it to the Silver Lab cluster (Table 4.1) for simulation. The number of total cells in the simulation was fixed (e.g. at 50,000) and the ratio of mossy fiber terminals, granule cells and Golgi cells was set using experimentally determined values for the density of each (6.6x10^5, 1.90x10^6 and 4607 per mm^3 respectively, data from Zoltan Nusser's lab). The thickness of the granule cell layer was set to 150μm , and the sides of the volume of granule cell layer chosen to give the assigned total number of cells with the correct densities in that volume. The connectivity conditions whereby granule cells seek the closest mossy fibre terminals etc. were removed in this network, as this did not impact simulation run time, and allowed very large scale networks to be quickly generated by neuroConstruct.

Figure 4.3A shows a 3D visualisation of an instance of the network with 50,000 cells. Two types of test were performed to investigate the scaling of the performance of this type of network across multiple processors, First, simulations containing the same numbers of cells were run across between 8 and 192 nodes (Figure 4.3B). The performance of 50,000 and 100,000 cell networks scaled approximately linearly with increasing numbers of processors. A second test of scaling was performed by keeping the total number of cells per processor identical. Figure 4.3B illustrates how the total time of simulation is approximately equal as the total number of processors grows, up to a simulations containing one million cells over 200 processors.

Figure 4.4A shows the second network used for parallel performance tests. This is an extension to 3D of the thalamocortical network of Traub et al (2005) (discussed in more detail in section 5.11). It consists of multiple layers of the cortex with five types of pyramidal cells, layer 4 spiny stellate cells and six types of interneuron. For the simulations run here, the thalamic input was disconnected and inhibition reduced to reproduce the network configuration of Figure 6 of Traub et al (2005). A Python script was used to generate the network in different sizes, scaling each population in the ratio used in the original 3,560 cell network. Figure 4.4A shows the network with

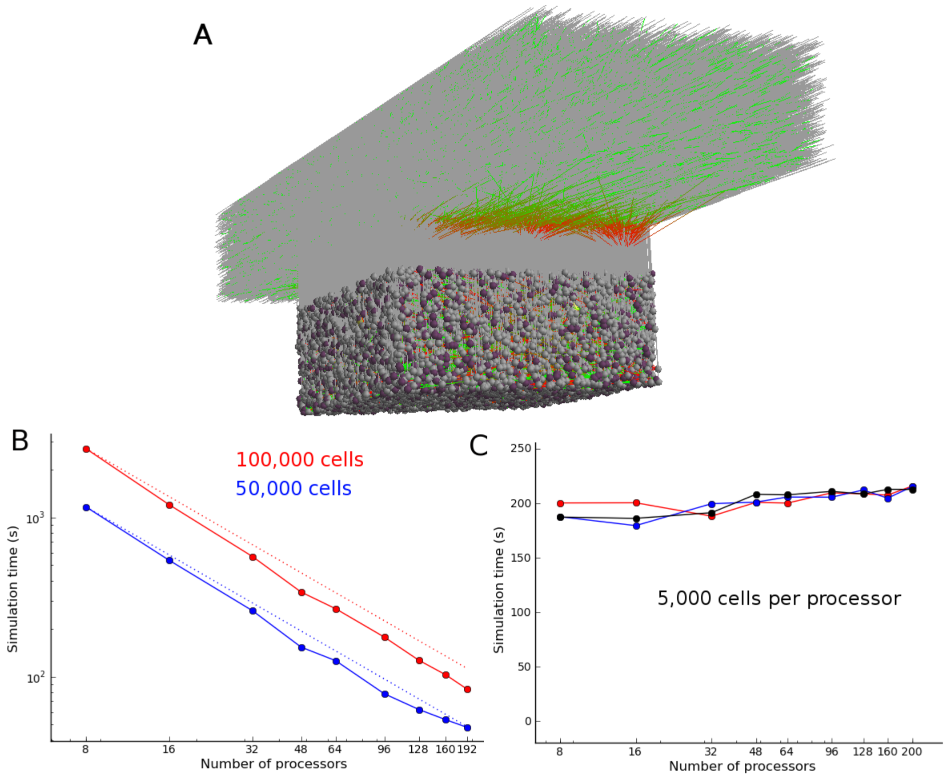


Figure 4.3: Parallel simulation of cerebellar granule cell layer. A) 3D visualisation of network containing 50,000 cells at realistic densities, consisting of 37,044 granule cells with parallel fibres (grey), 12867 mossy fibre terminals (purple) and 89 Golgi cells (yellow) in a 360.5μm x 360.5μm x 150μm volume of the granule cell layer. Synaptic connections are illustrated with lines changing from green (presynaptic location) to red (postsynaptic). B) Performance of networks of two sizes as number of processors is increased (circles). Dotted lines show ideal speed up of simulations from 8 processors. Simulation duration was 500ms, timestep was 0.025ms. C) Performance of network as number of cells per processor is kept constant. The three traces represent different seeds for generating network structure in neuroConstruct. Times represent duration of numerical simulation. Times for generation of scripts, simulation setup, and saving of results is not included.

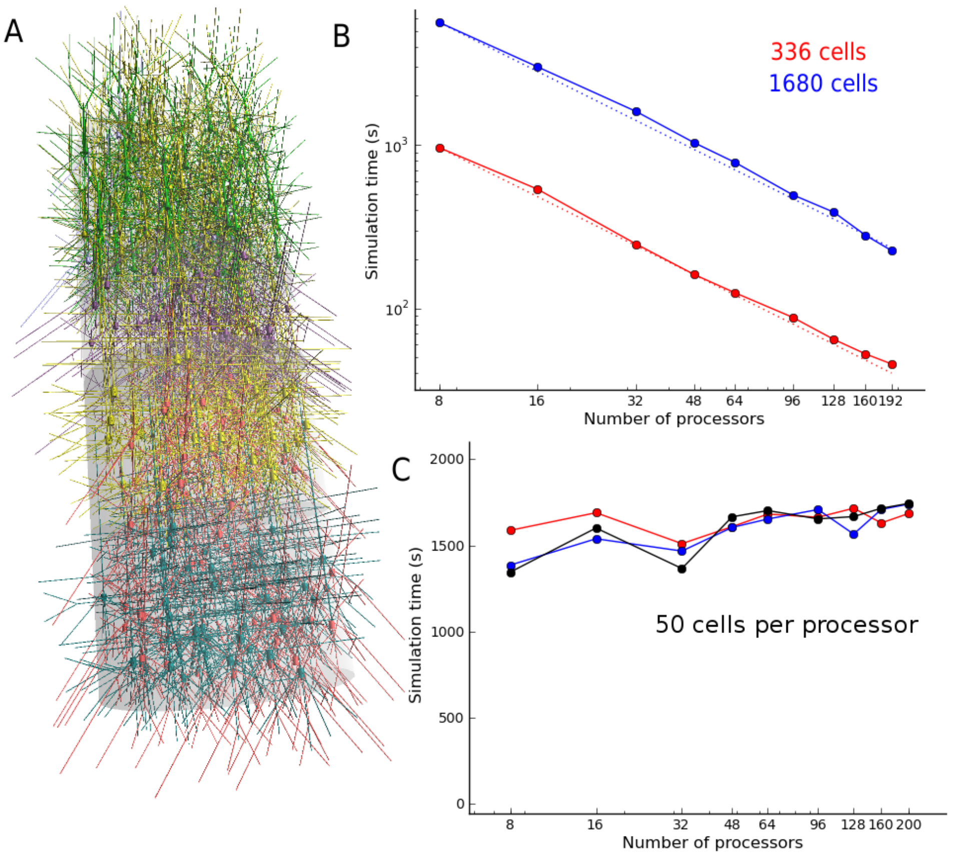


Figure 4.4: Parallel simulation of cortical column generated by neuroConstruct. A) Visualisation of network model consisting of 336 cells. Cell types present in the various cortical layers are outlined in section 5.11. B) Simulations of 336 and 1680 cells run across varying numbers of processors (circles). Dotted lines show ideal speed up of simulations from 8 processors. Simulation speed decreases almost linearly with number of processors. Simulation duration was 500ms, timestep was 0.025ms. C) Performance of network simulations of increasing size with the same number of cells per processor.

10% of the full network, and Figure 4.4B shows the performance of a 10% and 50% scale network over multiple processors, showing near linear speedup of the simulations as the total number of processors increases. Figure 4.4C illustrates performance of the parallel simulations when the number of cells per processor is held constant and the network size increased. This also indicates that the total simulation time is relatively independent of total network size, up to a network consisting of 10,000 detailed cells.

4.2.6 Limitations

While the parallel code generation of neuroConstruct can be used for a range of network sizes and complexities of cell models, there are a number of limitations with the current implementation. The round robin algorithm to assign cells to nodes requires a larger number of cells of each type compared to numbers of nodes for even load balancing. This may not be the case with, for example a cerebellar network with a few large Purkinje cell models and many smaller granule cells. This could be addressed in the future with support for NEURON's function to split large cells across multiple compute nodes (Hines et al, 2008c).

Another bottleneck for large scale simulations would be when the network connections contain many conditional connections dependent on distance between pre and post connection point. The time to generate such connections for a cell group of size N is of the order N^2. I have implemented some optimisations within neuroConstruct to improve the speed of network generation for single compartment cells, but more needs to be done for multicompartmental cells (e.g. ruling out cells from network connections based on intersoma distance before computing distances between dendrites and axons, etc.). These optimisations will be made internally in neuroConstruct, and the automated tests (section 4.5) will be used to ensure correct behaviour of network generation.

4.3 Dealing with the data deluge: compact data representations

A number of solutions have been developed to handle the large volumes of data associated with large scale network simulations. These have been based on HDF5, a technology which is widely used for compact data storage in both academia and industry. This binary storage format and an associated set of libraries and tools for reading/writing/visualising the contents of files is currently being developed by The HDF Group[12]. Data sets of any size can be stored in these files in a hierarchical structure, and the libraries provided can be used for rapid access to (sections of) this data.

4.3.1 Network representations

As mentioned in section 3.3.2 and 4.2.3 an equivalent version of the instance based form of NetworkML has been developed for HDF5, in addition to the standard XML format. This will store the cell positions, pre and postsynaptic connection locations, and external input information as structured binary information. This removes much of the overhead produced by the repeated tags in XML files (e.g. <***instance***>, <***location***>, <***connection***>). The dedicated libraries for HDF5 also mean that reading and writing HDF5 is much faster than the equivalent structures in XML. An example of a network stored in HDF5 format and viewed using the HDFView application[13] is shown in Figure 4.5. A comparison of the sizes of XML and HDF5 files for

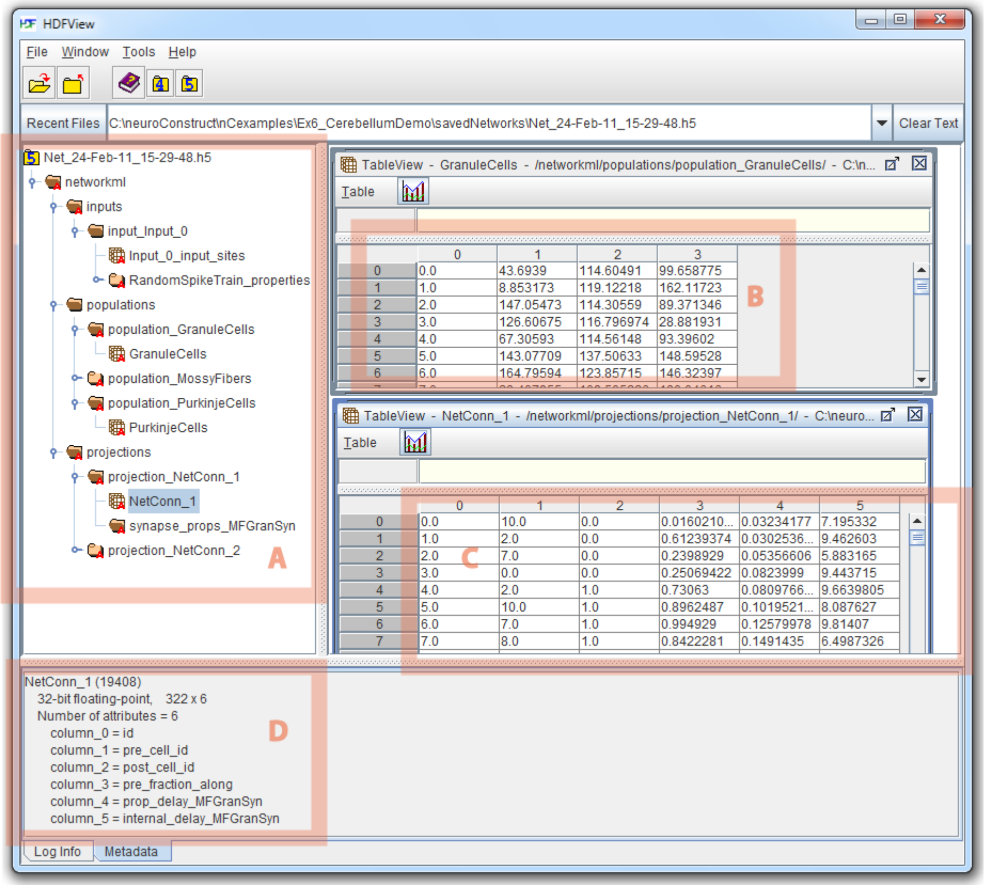
[12]http://www.hdfgroup.org
[13]http://www.hdfgroup.org/hdf-java-html/hdfview/

storing networks of different sizes is shown in Figure 4.6. The HDF5 files are consistently an order of magnitude smaller than the equivalent XML files.

As mentioned in section 3.5.4, neuroConstruct can save and reload network structures in both XML and HDF5 formats (either self generated or from another application supporting NetworkML). While lists of cell positions and connectivity are crucial for exchanging precise network structure data between applications, network generation templates are also useful as "recipes" for compact description of the network structure and for generating multiple instances of networks based on the same high level properties. neuroConstruct has its own internal format for generating 3D cell positions and network connections (section 2.2.3). These descriptions are stored in a few lines of the project configuration file and network instances are generated from these based on a seed. In an investigation involving multiple instances of networks of thousands of cells, the generated network can be stored, but much more efficient would be recording the generation algorithms and seeds used.

4.3.2 Simulation data sets

There is currently functionality present in neuroConstruct to save simulation data in HDF5 format. While text files containing lists of continuous membrane potential values or spike times are the default option for saving data, simulators supporting Python can easily save their simulation data in this much more compact binary format. Reloading of simulations into neuroConstruct when the data has been saved in HDF5 format is handled in the same way from the user point of view. Table 4.2 compares the size of data files produced when saving in text files or HDF5. Two possibilities are shown, saving the full voltage traces generated during the simulation



C:\neuroConstruct\nCexamples\Ex6_CerebellumDemo\savedNetworks\Net_24-Feb-11_15-29-48.h5

```
Net_24-Feb-11_15-29-48.h5
  networkml
    inputs
      input_Input_0
        Input_0_input_sites
        RandomSpikeTrain_properties
    populations
      population_GranuleCells
        GranuleCells
      population_MossyFibers
      population_PurkinjeCells
        PurkinjeCells
    projections
      projection_NetConn_1
        NetConn_1
        synapse_props_MFGranSyn
      projection_NetConn_2
```

TableView - GranuleCells - /networkml/populations/population_GranuleCells/ - C:\n...

	0	1	2	3
0	0.0	43.6939	114.60491	99.658775
1	1.0	8.853173	119.12218	162.11723
2	2.0	147.05473	114.30559	89.371346
3	3.0	126.60675	116.796974	28.881931
4	4.0	67.30593	114.56148	93.39602
5	5.0	143.07709	137.50633	148.59528
6	6.0	164.79594	123.85715	146.32397

TableView - NetConn_1 - /networkml/projections/projection_NetConn_1/ - C:\neuro...

	0	1	2	3	4	5
0	0.0	10.0	0.0	0.0160210...	0.03234177	7.195332
1	1.0	2.0	0.0	0.61239374	0.0302536...	9.462603
2	2.0	7.0	0.0	0.2398929	0.05356606	5.883165
3	3.0	0.0	0.0	0.25069422	0.0823999	9.443715
4	4.0	2.0	1.0	0.73063	0.0809766...	9.6639805
5	5.0	10.0	1.0	0.8962487	0.1019521...	8.087627
6	6.0	7.0	1.0	0.994929	0.12579978	9.81407
7	7.0	8.0	1.0	0.8422281	0.1491435	6.4987326

```
NetConn_1 (19408)
  32-bit floating-point,   322 x 6
  Number of attributes = 6
    column_0 = id
    column_1 = pre_cell_id
    column_2 = post_cell_id
    column_3 = pre_fraction_along
    column_4 = prop_delay_MFGranSyn
    column_5 = internal_delay_MFGranSyn
```

Figure 4.5: A screenshot of the HDFView application showing the contents of a NetworkML file saved in HDF5 format. The hierarchical structure of the file can be seen (A, compare to structure of NetworkML elements in Figure 3.6). Cell positions are stored in separate arrays per cell group consisting of rows of floating point values for cell index and (x,y,z) location (B). Similarly information on synaptic connections is stored in arrays with one row per pre- to postsynaptic connection (C). The values stored in each column are specified in the metadata (D). Connections between single compartment cells, or where no synaptic delay is present, can be saved in arrays with fewer columns, so reduce the size of the file.

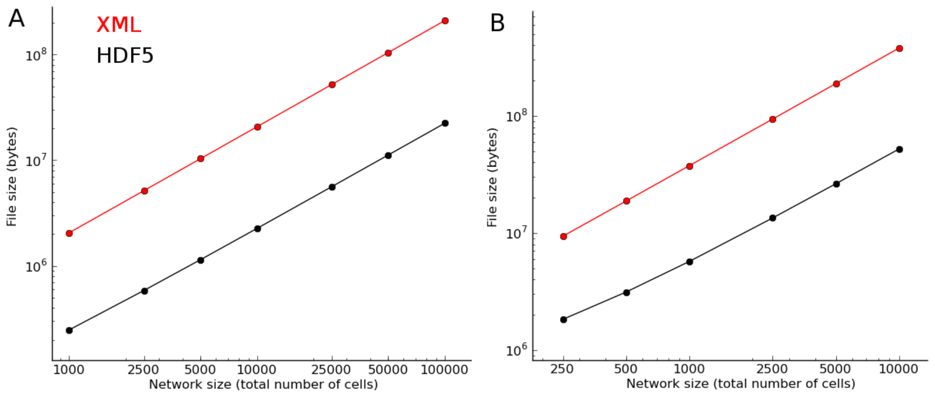


Figure 4.6: Comparison of file sizes for NetworkML in different formats. A) Size of NetworkML files for granule cell layer network (as used in Figure 4.3) of different sizes. Number of synaptic connections is approximately 6.7 times the total number of cells. B) Size of NetworkML files for thalamocortical network model (as used in Figure 4.4) of different sizes. Number of synaptic connections is approximately 183 times the total number of cells

for every cell, or just saving spike times. The binary format offers advantages when saving the full data, both in overall data size and number of files. While the spike time data is more compact in text format, the smaller number of files generated by the HDF5 option is convenient from a data management point of view.

More options are being explored to further improve the storage of simulation data. HDF5 libraries have internal data compression algorithms which can reduce overall size of files, albeit making reading and writing slower. Parallel HDF5 offers the opportunity to save to a single HDF5 file from distributed processes, e.g. cells in a population spread across nodes. This would further reduce the number of simulation data files produced.

A number of other applications use HDF5 as a data storage format for simulation results (e.g. PyNN, Neuronvisio[14], NeuroTools[15] and Neu-

[14]http://mattions.github.com/neuronvisio
[15]http://neuralensemble.org/trac/NeuroTools

Network model	**Text files Full data**	**HDF5 Full data**	**Text files Spike times**	**HDF5 Spike times**
Granule cell layer	790 MB 5000 files	400 MB 24 files	0.24 MB 5000 files	0.66 MB 24 files

Table 4.2: Simulation data set sizes. The simulation of a granule cell layer model (section 5.1.4; Figure 4.3) with 5,000 cells was run for 500ms with 0.025ms timestep over 8 processing cores. Data was saved of either the full membrane potential traces or just the spike times, in either text file format, or HDF5. There are 3 cell groups in the network and a single HDF5 file is generated for each of these on the processors.

roHDF[16]). neuroConstruct can import time series data from a wide range of HDF5 based data files (section 2.3.2). This functionality is available through the GUI (File → Import → Import Data for Plot: HDF5 File). This functionality can also be accessed through the Python interface, making it easier to analyse and compare large data sets.

4.4 Development of a Python based scripting interface

4.4.1 Python in computational neuroscience

More and more researchers with formal training in computer science or professional programming experience are entering the field of computational neuroscience. This has lead to a movement towards using well established software development practices (e.g. version control, open source development philosophies) and a desire to use high quality, widely used, freely available packages whenever possible, as opposed to coming up with domain specific solutions. This has lead to the increased use of XML and HDF5 for data storage, MPI for parallel network simulations, as well as the Python language as a scripting interface for a number of computational neuroscience

[16]https://github.com/unidesigner/neurohdf

tools.

Python is an interpreted programming language which incorporates a number of features required in modern scripting languages. It has a clear and expressive syntax making it easy to learn, write and read, uses object oriented programming conventions, contains a large number of standard library modules, is easy to extend with new modules, works transparently across all major operating systems and can be interfaced with packages in other languages such as C++ and Java. There is a large user community developing both the language itself and the large number of extension modules including many specifically for scientific computing (e.g. NumPy).

Python as a general purpose language for scientific computing is often compared to MATLAB. It is as just as easy to learn, but has the distinct advantage that the core platform along with almost all of the extension modules are freely available and open source. This is advantageous not only for a developer creating scripts for private use, but also increases the chances that others will reuse any utilities or applications developed.

Python is becoming the interface language of choice for a number of tools in neuroscience (Davison et al, 2009). Many simulators have recently had Python scripting interfaces added, including NEST (Eppler et al, 2008), NEURON (Hines et al, 2009), MOOSE (Ray and Bhalla, 2008), Brian (Goodman and Brette, 2008), NCS (Drewes et al, 2009), STEPS (Wils and De Schutter, 2009), PCSIM (Pecevski et al, 2009) and Topographica (Bednar, 2009). This spread of Python is in part due to a desire by developers to concentrate their efforts on the core added value features of their applications as opposed to creating/maintaining/documenting an application specific scripting interface.

4.4.2 Implementation in neuroConstruct using Jython

Jython[17] is an implementation of the Python programming language in Java (the most widely used version of Python is written in C). This allows most Python programs which use the core Python modules to run without modification on Jython, but significantly, Jython scripts can import libraries written in Java. This has allowed all of the functionality in the neuroConstruct source code to be accessed from within a Python based script. Such a script can load a neuroConstruct project, make changes to the settings, generate scripts for a given simulator and execute them, all in a few lines of code (Figure 4.7).

Jython support is built in to the standard release of neuroConstruct, and any script requiring the neuroConstruct core functionality can be easily executed using the ***nC*** utility:

Linux/Mac: ***./nC.sh -python MyScript.py***
Windows: ***nC.bat -python MyScript.py***

Some of the scenarios in which developing scripts for the Python interface would be a significant advantage over interacting with project through the GUI include:

- **Model behaviour under different inputs:** Generating frequency vs. input current (or synaptic input firing rate) curves from cell models is a common way to investigate their behaviour and compare to experimental measurements.
- **Robustness testing:** Systematically altering ill constrained model parameters (e.g. channel conductance densities) and investigating changes in firing behaviour.

[17] http://www.jython.org

```
import time
                                                                          A
from java.io import File

from ucl.physiol.neuroconstruct.project import ProjectManager
```

```
pm = ProjectManager()                                                     B
project = pm.loadProject(File("Parallel.ncx"))
```

```
project.cellGroupsInfo.getCellPackingAdapter("ExcCG").setMaxNumberCells(20)
project.cellGroupsInfo.getCellPackingAdapter("InhCG").setMaxNumberCells(10)   C
```

```
neuroConstructSeed = 1234
simConfig = project.simConfigInfo.getDefaultSimConfig()                   D
pm.doGenerate(simConfig.getName(), neuroConstructSeed)

while pm.isGenerating():
    time.sleep(1)
```

```
simulatorSeed = 1234
project.genesisFileManager.generateTheGenesisFiles(simConfig,             E
                                                   simulatorSeed)
```

```
success = pm.doRunGenesis(simConfig)
                                                                          F
print "Sucessfully ran GENESIS: %i" % success
```

Figure 4.7: This script loads a neuroConstruct project (e.g. previously constructed and saved through the GUI), changes the number of cells in a population, and generates scripts for running in the GENESIS simulator. A) The first part of the file imports a number of modules including native Python (***time***), standard Java classes (***java.io.File***), and a class from the neuroConstruct Java code. B) A ***ProjectManager*** object is created and the neuroConstruct project, Parallel.ncx, is loaded. C) The variable ***project*** is a ***Project*** object containing details of the network to be generated and ***cellGroupsInfo*** in this is used to alter the number of cells in 2 Cell Groups. D) The seed for generating the random positions and connectivity of the network is set, the default simulation configuration (section 2.1.3) is selected, and the network is generated. A ***while*** loop has been inserted to wait until the network is generated before proceeding. E) The GENESIS files are generated. F) The GENESIS files are set running. These execute in a separate process and so at this point the script could exit or reset more parameters and generate a new simulation.

- **Custom algorithm for tuning model to experimental data:** The Python interface allows the flexibility to run a simulation, reload results, compare to target data, adjust the model, and run again. In this way, complex algorithms can be created for tuning models to match experimental data.
- **Generating populations of models:** Most cell and network simulations will contain a degree of randomness, either in the stochasticity of the inputs or of inhomogeneity in network structure and properties, meaning each simulation will produce slightly different results. The Python interface is an ideal way to manage the large number of simulations which need to be generated when analysing such models.
- **Use of subset of neuroConstruct functionality:** A specific feature of neuroConstruct can be used in isolation, e.g. conversion from one morphological format to another, converting a ChannelML file into a mod file and compiling, etc. In this way neuroConstruct can be just one element in a larger toolchain linked by Python.

A set of utility functions for controlling neuroConstruct projects is included with the standard release (***ncuils.py***). This includes the ***SimulationManager*** class which can be used to run multiple simulations across simulators with a single function call, and a function to facilitate generation of input current versus firing frequency plots (used to generate the plots in Figures 5.2 and 5.3).

The parallel simulation generation functionality of neuroConstruct can also be accessed via the Python interface. The parallel configuration to use can be set by specifying which parallel configuration (***MPIConfiguration*** class) to use. This functionality was used to generate the plots in Figures

4.3 and 4.4. Useful analysis scripts in Python can be distributed with neuroConstruct projects, and many examples are available for the core example projects (chapter 5) in their ***pythonScripts*** directories.

One of the drawbacks of using Jython for the scripting interface of neuroConstruct is that the Java implementation often lags behind the more widely used C implementation of Python, and has a smaller developer base. Another disadvantage is that many Python packages which link to core functionality in C++ libraries (e.g. matplotlib for plotting) are not accessible in Jython scripts. However, these disadvantages are more than compensated for by the ability to access any Java functionality in neuroConstruct without any extra effort, using a modern scripting interface.

4.5 Automated testing infrastructure

Automated code testing is a crucial part of modern software development. Both unit testing, where individual methods of the source code are tested, and system testing, where a sequential set of high level functions are tested together, are practised, usually within a framework where such tests can be routinely executed during the development cycle to ensure that no new code introduces bugs into the system.

A number of such frameworks are available to software developers, usually closely tied to the programming language being used. Apache Ant[18] is a widely used, Java based build process tool which allows automation of tasks such as compiling code, generating documentation, testing and running applications. This was chosen for automating testing as the neuroConstruct build process was already based on Ant. In addition to building the main Java jar file (which can be done on any operating system by typing ***ant***

[18]http://ant.apache.org

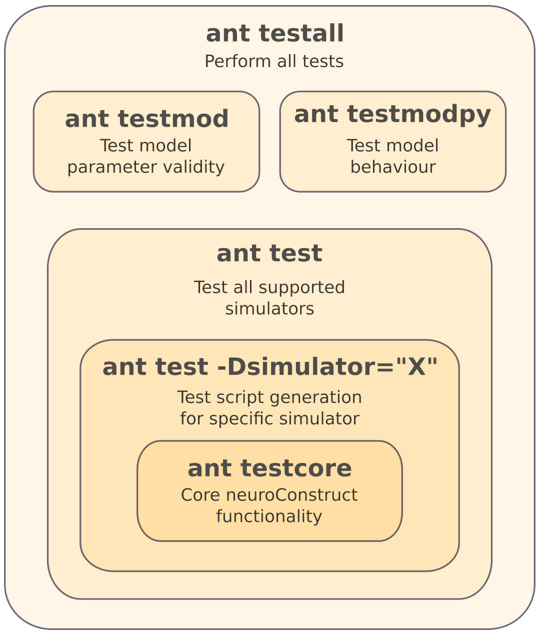

ant testall
Perform all tests
ant testmod
Test model parameter validity
ant testmodpy
Test model behaviour
ant test
Test all supported simulators
ant test -Dsimulator="X"
Test script generation for specific simulator
ant testcore
Core neuroConstruct functionality

Figure 4.8: Hierarchy of tests which can be performed on neuroConstruct and associated models using Apache Ant. These can be used to test specific functionality within neuroConstruct itself, check the validity of the model structures and parameters in the key example projects, or execute the models on multiple simulators, comparing the activity of the cells on each against expected model behaviour

jar) and running neuroConstruct (***ant run***), a number of Ant tasks were created to test the application and the core model examples at various levels (Figure 4.8).

A test suite has been implemented for the core functionality of neuroConstruct (***ant testcore***). This runs unit tests on all the main classes such as those for representing cell structures, and tests generated network properties against the template parameters used to create them. At a level higher than this, interaction with supported simulators can be tested (***ant test***, with an optional ***-Dsimulators*** flag to test only selected simulators). These basic tests check that the simulator scripts are correctly generated (and mod files etc. are compiled), that the simulators can be launched, and that simulation data is saved in the correct format.

In addition to checking neuroConstruct itself, tests have been created for the core set of models which are distributed with neuroConstruct (including all those described in chapter 5). A number of tests can be performed on the properties of the cells, channels, and networks stored in these projects (***ant testmod***), consisting of checking the validity of the NeuroML components, checking morphological properties of the cells, and basic checks on sufficient annotation/comments in the project. Many of these checks are the same as those run when the Validate button is pressed in the neuroConstruct GUI.

To check that the models are producing expected behaviour, a longer set of tests were created (***ant testmodpy***) which involve generating the projects, creating scripts for a number of simulators, executing them, and testing model behaviour against an expected data set. This check ensures that models behave identically on supported simulators[19] and that any updates to neuroConstruct or any of the supported simulators do not change model behaviour. A final test (***ant testall***) performs all tests in sequence and has proven useful for final overall checks before new releases of the software.

This hierarchical testing structure is invaluable for quality control of the application and the associated models. Optimisations in the core applications are often implemented to speed up network generation or simulation data handling, and it is useful to have an easy way to test this new functionality quickly with a wide range of models. As new models get added to the core too, this infrastructure can be used to encourage minimum standards of project validity and testability.

[19]The tests usually consist of making sure spike times are identical to within a small tolerance, normally less than 0.1 ms in simulations of a few hundred milliseconds.

4.6 Interacting with subcellular signalling pathways

Subcellular signalling pathways can play an important role in information processing in neurons, especially at synapses, and the systems biology community has been produced many detailed models of these pathways (Kotaleski and Blackwell, 2010). Resources like the BioModels Database and the CellML Model Repository contain hundreds of models which could be of benefit for computational neuroscientists, not only related to the electrical behaviour of neurons, but also other cellular level models and metabolic processes which can be important for brain function. These communities have been quite independent until recently however, and there has been little exchange of models or technical solutions between the fields.

In order to facilitate the integration of such models into conductance based neuronal simulations (which would allow multiscale modelling across levels I-VI in Figure 1.1) I have implemented an SBML import function for libNeuroML (Figure 3.17). This uses the Java API for parsing SBML, JSBML[20] to read in SBML files and convert them to ComponentClass definitions in LEMS (section 3.6.3). Once a model is in this format, it can be used by the LEMS Interpreter application, or mapped to one of supported export formats from libNeuroML.

The example shown in Figure 4.9 was retrieved as a valid SBML file taken from the BioModels database[21]. It is a model of spontaneous Ca^{2+} oscillations in astrocytes (Lavrentovich and Hemkin, 2008) and features Ca^{2+} induced Ca^{2+} release from the endoplasmic reticulum, and feedback via cytosolic IP_3. Figure 4.9A shows a schematic of the model generated from

[20]http://sbml.org/Software/JSBML

[21]http://www.ebi.ac.uk/biomodels-main/BIOMD0000000184

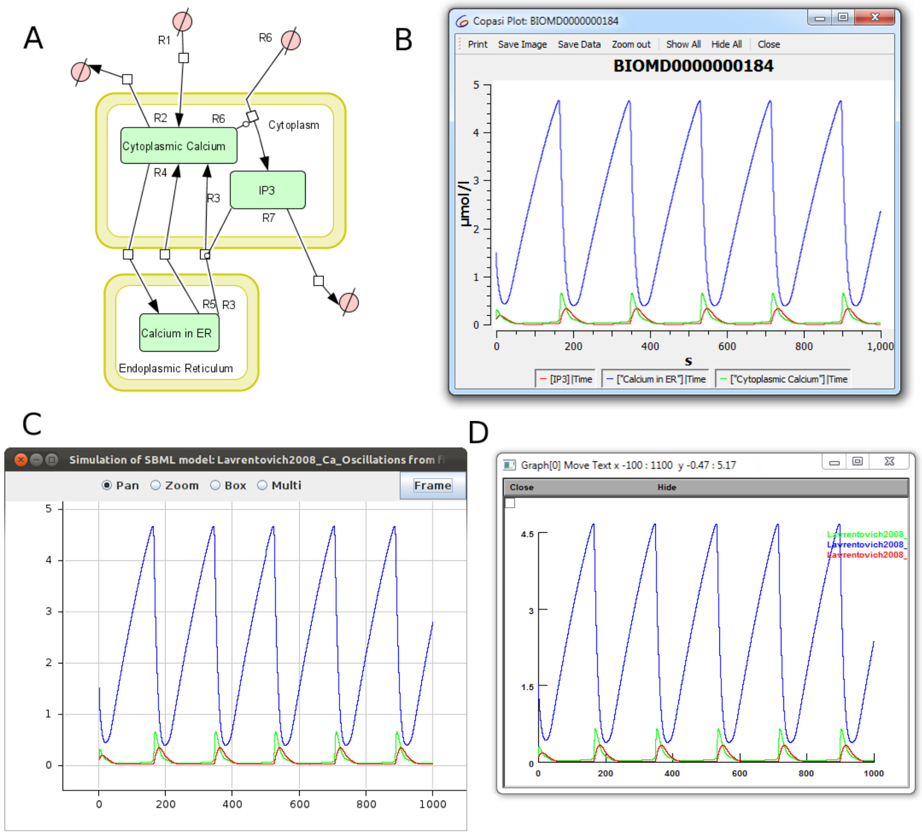


Figure 4.9: Example of SBML model run across multiple simulators. A) Schematic of an SBML model of spontaneous Ca^{2+} oscillations in astrocytes. The core SBML entities compartments (yellow outlines), species (green boxes) and reactions (black arrows) are depicted in the diagram, which was automatically generated from the XML file by CellDesigner. B) Simulation of the model by Copasi. Traces show $[Ca^{2+}]$ in the cytoplasm (green), endoplasmic reticulum (blue) and cytosolic IP_3 (red). C) Model behaviour after importing into libNeuroML and execution with libNeuroML. D) Simulation generated for NEURON by libNeuroML. Traces in C and D as in B.

the SBML file by CellDesigner[22], a widely used application for editing, simulating and generating graphical representations of SBML models. Figure 4.9B shows the behaviour of the model when executed in Copasi[23], another popular SBML compliant application. This illustrates the oscillatory nature of the cytosolic Ca^{2+} in the model. The SBML file was loaded into libNeuroML, and a single Component containing an instance of the model was created, and the behaviour simulated with the LEMS Interpreter (Figure 4.9C). Once the model was in LEMS it could also be mapped to NEURON. An NMODL file was generated with state variables representing the species present in the original SBML file. A NEURON section was created with the mechanism, and a simulation of this was run (Figure 4.9D).

The numerical integration method used in the LEMS Interpreter is quite basic (forward Euler), but there is still good agreement between traces produced by it and dedicated SBML simulators. The functionality for importing SBML files was also tested against the SBML Test Suite[24]. This represents a collection of SBML files covering all of the main features in the specification, along with analytically generated solutions to model behaviour and is intended for benchmarking SBML compliant applications, and checking compliance to the standard. Currently, libNeuroML, importing the test suite models and executing them with the LEMS interpreter can pass 331 out of 952 tests[25].

The support for SBML import will be expanded in future, along with testing of generated code on other libNeuroML export formats such as Brian and NEST. The existing functionality however is a major step towards incor-

[22]http://www.celldesigner.org

[23]http://www.copasi.org

[24]http://sbml.org/Software/SBML_Test_Suite

[25]Quite a number of the tests are for edge cases of the specification, and to date only one SBML application claims to pass all tests

porating such models into dedicated computational neuroscience simulators. An initial implementation of export of SBML compliant files from libNeuroML has also been developed. This will be limited however to single instances of simple cells as there is currently no representation for populations of models or partial differential equation support (for multicompartmental cells) in SBML. Support for importing and exporting CellML models has also been started, but is in an early stage of development.

4.7 Conclusions

Large scale models of the nervous system can be used for investigating the behaviour of networks of neurons at close to anatomical numbers, driven by realistic levels of simulated synaptic input. The basis for macroscopic effects such as local field potentials can be studied, as well as the spatial spread of correlated activity, which can be related to experimental measurements. Software solutions for creating such simulations are being developed, and it will be important to allow these to be used by as wide a range of experimental neuroscientists as possible to ensure the anatomical and physiological realism of the models.

Automated generation of scripts for Parallel NEURON by neuroConstruct is a significant step in this direction. Simulations can be generated and managed from the application and run on remote high performance computing resources. Parallel simulations consisting of 10,000 detailed cells and one million simple cells have been tested. Solutions have also been developed for saving network structure and simulation data in more compact binary formats, to deal with the large data sets such simulations bring.

The Python interface neuroConstruct gives advanced users of the application powerful, low level access to all of the core functionality of the application. This allows scripts to be created for generating and analysing large families of models, to systematically probe the behaviour of cell and network models. The automated testing framework for neuroConstruct and the associated example projects ensures that updates to the application core or changes to models do not lead to unexpected behaviour.

Greater integration with SBML and systems biology languages will allow a host of new model components from these initiatives' model repositories to become available for integration into neuronal models. This will enable true multiscale models to be created incorporating gene regulation, chemical and electrical signalling, morphological detail and 3D positioning and connectivity. This interoperability will also encourage greater interaction between the members of the computational neuroscience and systems biology communities for both biological investigations and technical issues.

Chapter 5

Biophysically Detailed Cell and Network Models

In this chapter I describe a number of the cell and network models which have been imported for use in neuroConstruct, the majority of which are also available as fully compliant NeuroML. These have mainly been based on published modelling studies, where the simulation scripts had been made publicly available. Table 5.1 gives an overview of the models which are described in this section.

Many of these were converted as proof of principle of the ability of NeuroML to represent detailed cell and network models of a variety of cells from different sources, and some of these have been extended to utilise the functionality enabled by a 3D representation in neuroConstruct. The models developed/extended in the Silver Lab for original scientific research are also featured.

As the majority of the models are based on cells and microcircuits of the cerebellum and neocortex, I will discuss the anatomy of these regions briefly and mentioned some of the work done in creating computational models of their cells and networks.

Brain region	**Source of model**	**Cell Types**
Cerebellum	Maex and Schutter (1998)	Granule cell Golgi cell
	Volker Steuber and Chiara Saviane, based on Berends et al (2005)	Granule cell
	Vervaeke et al (2010)	Golgi cell
	Solinas et al (2007a,b)	Golgi cell
	De Schutter and Bower (1994)	Purkinje cell
Neocortex	Mainen et al (1995)	Layer 5 pyramidal cell
	Kole et al (2008); Rothman et al (2009)	Layer 5 pyramidal cell
	Traub et al (2005)	Layer 2/3 Regular Spiking (RS) pyramidal cell Layer 2/3 Fast Rhythmically Bursting (FRB) pyramidal cell Superficial basket cell Superficial Axo-axonic cell Superficial Low Threshold Spiking (LTS) interneuron Layer 4 Spiny stellate cell Layer 5 Tufted Intrinsically Bursting (IB) pyramidal cell Layer 5 RS pyramidal cell Deep basket cell Deep Axo-axonic cell Deep LTS interneuron Layer 6 pyramidal cell
Thalamus	Traub et al (2005)	Thalamocortical relay cell Nucleus reticularis thalami cell
Hippocampus	Migliore et al (2005)	CA1 pyramidal cell
Dentate Gyrus	Santhakumar et al (2005)	DG Granule cell Hilar perforant-path associated (HIPP) cell Mossy cell Basket cell

Table 5.1: Cell and network models converted to neuroConstruct/NeuroML format

5.1 Cerebellar models

5.1.1 Cerebellar anatomy and physiology

The cerebellum has been a favourite brain region for theorists interested in how the nervous system processes information for over a century. There are many reasons for this including the relatively small number of identifiable cell types, the repeated patters of cell layout and connectivity suggesting the presence of a canonical circuit, and the fact that cerebellum like structures are present in most classes of vertebrates (Bell et al, 2008), suggesting it plays key role in the nervous system for higher animals.

The histological investigations of Ramon y Cajal and Golgi (D'Angelo et al, 2011) revealed the distinct classes of cells present in the cerebellum and particularly the remarkable innervation of the parallel fibres on the planar dendrites of the Purkinje cells. Their studies laid down the main connection pathways between the cell classes, and made the distinction between cells whose axons projected out of the structure, and so were likely to have an influence on motor activity, fibres emanating from outside the cerebellum which could carry sensory information, and those cells whose axon and dendrites are restricted to the cerebellar cortex and so were involved in local computation. The involvement of the cerebellum in motor coordination was well established at the time from studies of cerebellar ablation in dogs and monkeys, studies of patients with cerebellar tumours and further work studying soldiers with brain injuries during the first World War (Holmes, 1917) laid a firm basis for the clinical descriptions of cerebellar dysfunction.

Work to fully describe the cerebellar circuitry continued and a significant step came in the 1960s and 1970s with the publication of a number of books on anatomical and physiological properties of the cerebellum (Eccles et al,

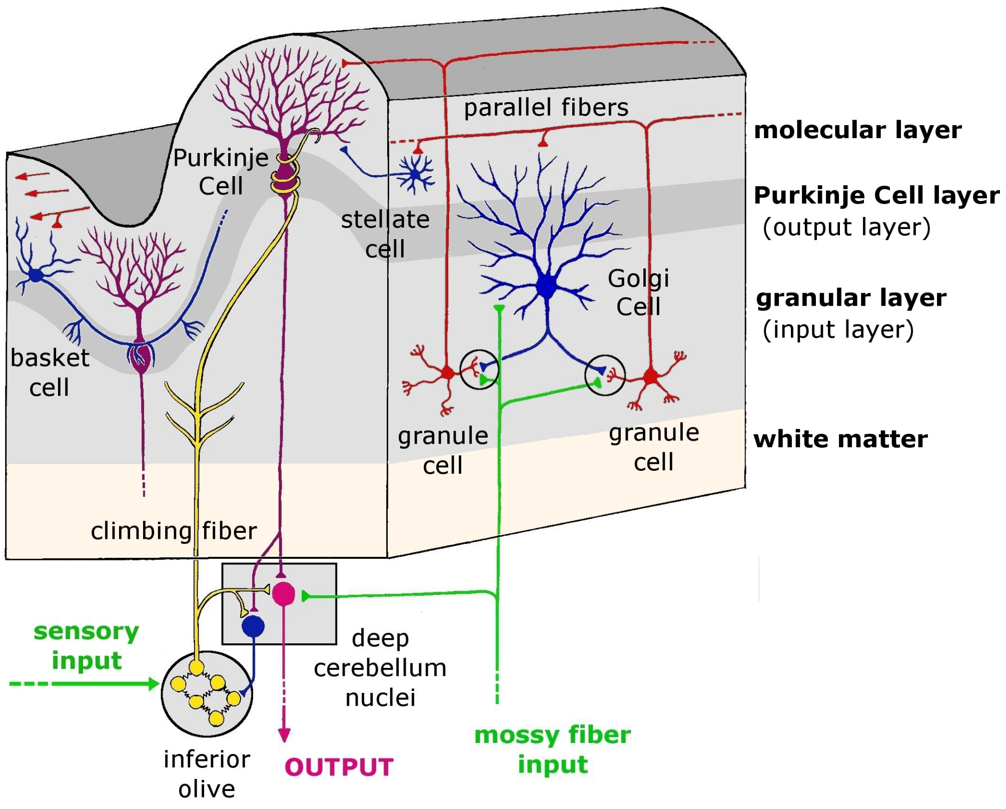


Figure 5.1: Diagram of olivo-cerebellar system showing main cell types. The cerebellar cortex contains three main layers: the granular layer with mossy fibre terminals, granule cells and Golgi cell somata; the Purkinje cell layer consisting of a one soma thick layer of Purkinje cell bodies; and the molecular layer, with Purkinje cell dendrites, granule cell axons forming the parallel fibres, basket and stellate cell interneurons. The inferior olive projects its climbing fibres to the cerebellar cortex, targeting Purkinje cell dendritic trees. The deep cerebellar nuclei receive the sole output of the cerebellar cortex, inhibitory synapses from the Purkinje cells. See text for more details. Figure courtesy of Matteo Farinella (Silver Lab).

1967; Palay and Chan-Palay, 1974). Work has progressed and now there is consensus on the main cell classes, and connectivity within the structure. The main cell types present in the cerebellar cortex and associated sub-cortical structures are shown in Figure 5.1. The cerebellar cortex receives input via two pathways: sensory and motor control input via the mossy fibres and from the inferior olive via the climbing fibres. The mossy fibres enter the cerebellar cortex and form synaptic connections with granule cell dendrites in glomeruli in the lower granular layer. The granule cells, which are the most numerous cell type in the mammalian brain, gather input from approximately four mossy fibres and extend long axons into the molecular layer, where they bifurcate and extend up to 2.5mm in each direction along the medio-lateral axis of the cerebellar cortex (Harvey and Napper, 1988). The parallel fibres form en-passant excitatory synapses on the almost flat dendritic trees Purkinje cell arranged in parallel in the parasagittal plane (Eccles et al, 1967). These complex cells form the sole output of the cerebellar cortex, and receive directly the other main cerebellar input, the climbing fibres. While Purkinje cells receive synaptic input from $\sim$175,000 different parallel fibres (Napper and Harvey, 1988), a single climbing fibre makes extensive synapses on a Purkinje cell's dendritic tree, reliably driving spiking in the target cell.

Interneurons of the cerebellar cortex include the Golgi cell with somata in the granular layer. These spontaneously active cells receive excitatory input from the parallel fibres through a dendritic tree ascending into the molecular layer. Their axons enter the glomeruli inhibiting the granule cells. Golgi cell also receive input directly from mossy fibres (Kanichay and Silver, 2008). Golgi cells form electrical synapses between their dendrites, allowing synchronisation (Dugué et al, 2009) and desynchronisation (Vervaeke et al, 2010) of Golgi cell firing. The two main classes of interneurons in the molecular layer are the basket cell and stellate cells. These receive excitatory input from the parallel fibres and inhibit the Purkinje cells; the stellate cells on their dendrites and the basket cells their somata and initial axon segment.

The deep cerebellar nuclei are the sole output of the olivo-cerebellar system. These receive inhibitory input from the Purkinje cells as well as some direct mossy fibre input. They also receive excitatory input from, and provide inhibition to cells in the inferior olive. These cells, which emit the climbing fibres are electrically coupled, and there is evidence that Purkinje cells with specific receptive fields send output to distinct regions of the cerebellar nuclei, and in turn receive input from climbing fibres of common origin, thus forming discrete microcomplexes (Apps and Garwicz, 2005).

Not included in the figure are the Lugaro cell, an inhibitory cell of the granular layer which receives mossy fibre input and inhibits Golgi cells, and the unipolar brush cell, an excitatory cell also found in the granular layer (particularly in the vestibulocerebellum), which is believed to provide feedforward excitation to granule cells (Dino et al, 2000).

5.1.2 Theoretical and computational models of cerebellar function

Models of the function of the cerebellum fall into a number of categories, but have generally been inspired by the unique cerebellar cytoarchitecture, particularly the spread of the parallel fibres through the molecular layer and their massive convergence onto the Purkinje cell dendritic tree. Braitenberg and Atwood (1958) suggested that the flow of activity along the parallel fibres (they assumed a slow conduction of impulses) allowed the cerebellum to transform spatial to temporal patterns of activation. Each Purkinje cell received input originating from mossy fibre activity at different times in the past, and its output depended on a complex transformation of this information.

The anatomical work of Eccles et al (1967) inspired theories by Marr (1969) and Albus (1971) which were based on learned pattern recognition. Both of these focussed on changes in the synaptic strength between the parallel fibre and Purkinje synapses as the location where motor patterns were stored. The divergence of the mossy fibre input onto a larger number of granule cells enables separation of similar input patterns, and the large number of synapses on the Purkinje cell allowed a wider range of patterns to be stored. The climbing fibre input acted as the teaching signal to change synaptic strength of the parallel fibre-Purkinje cell synapse. A key difference between the theories was that Marr believed that these synapses were only strengthened, whereas Albus believed they could also be weakened (he also allowed for alterations in the parallel fibre-molecular layer interneuron synaptic strengths). Experimental verification of long term depression (LTD) at this synapse helped the latter view (Ito et al, 1982).

Other types of models for understanding cerebellar function have been developed, including some based on Control Theory, where the cerebellum is thought to form an internal model of the motor apparatus to fine tune movements (Wolpert et al, 1998). Other researchers have concentrated on creating detailed conceptual and computational models of cerebellar information processing focussing on specific behavioural tasks, e.g. the role of learned timings in Pavlovian eyelid conditioning (Medina et al, 2000; Ohyama et al, 2002).

The Marr-Albus (or Marr-Albus-Ito) motor learning theory has proven a popular with cellular physiologists due to the use of detailed anatomical and physiological data in their formalisms. However, more recent experimental data has highlighted the shortcomings of the Marr-Albus view of cerebellar function (D'Angelo et al, 2011; Rokni et al, 2008). Synaptic plasticity is much more prevalent in the cerebellar circuitry than just at the parallel fibre to Purkinje cell synapse (Hansel et al, 2001), although Albus had predicted some other locations of plasticity. Oscillations in olivo-cerebellar networks suggest that this pathway could be involved in a temporal coding scheme, not just as a error signal source (Zeeuw et al, 2008). Localised stimulation of the granule cell layer produces more spikes in the Purkinje cells above the location of stimulation compared to the Purkinje cells along the parallel fibre (Cohen and Yarom, 1998; Rokni et al, 2008). These aspects all point to the need for models encapsulating the latest anatomical and biophysical detail of the constituent parts of the cerebellum to check the consistency of the various theories being proposed for its function.

A number of large scale cerebellar models have been implemented to computationally test these theories (Medina et al, 2000; Tyrrell and Willshaw, 1992). These generally use simplified representations of the individual cells, although with realistic connectivity and often synaptic plasticity properties. A number of biophysically detailed models of individual cerebellar cells have been produced (De Schutter and Bower, 1994; Molineux et al, 2005; Nieus et al, 2006; Solinas et al, 2007a; Steuber et al, 2011; Vervaeke et al, 2010) and these can be used as the basis for more network models containing greater anatomical and physiological realism (Berends et al, 2004; Gleeson et al, 2007; Howell et al, 2000; Solinas et al, 2010).

5.1.3 Cerebellar cell models

Granule cell models

Cerebellar granule cells are some of the smallest and simplest cell types in the central nervous system, but they make up almost half of all neurons in the brain. They are densely packed in the granular layer, are electrically compact and possess a small number of short dendrites ($\sim$4) connecting to the cerebellar glomeruli, where they receive excitatory input from mossy fibres and inhibition from Golgi cells. The axons of the granule cells form the parallel fibres, transmitting their activity to multiple Purkinje cells in the molecular layer.

Two granule cell models have been converted to NeuroML. The first of these is from a network model of the rat granule cell layer (Maex and Schutter, 1998). This single compartmental model was based on an earlier multicompartmental model of a turtle granule cell from Gabbiani et al (1994). As well as reducing the model to a single compartment, shifts were introduced in the voltage dependence of the channels to allow for the higher firing threshold of the rat granule cell. The conductances were provided by fast Na^+ channels, delayed rectifier K^+, A-type and $[Ca^{2+}]$ dependent K^+ channels, high voltage activated Ca^{2+}, and an anomalous inward rectifier. The model also contained a simple exponentially decaying Ca^{2+} pool. The response of the cell to current clamp injection is shown in Figure 5.2A.

The second granule cell model was developed by Volker Steuber and Chiara Saviane (Silver Lab). This was based on a published update of the previously mentioned granule cell model by Berends et al (2005). In addition to the channels in the above model, it included a persistent Na^+ channel, a slow K^+ channel and an inward rectifying non inactivating K^+ channel.

Channel kinetics and maximal conductances to retune the model to match experimental data obtained in the Silver Lab. The response of this cell to current clamp injection is shown in Figure 5.2B. Input current versus firing rate for these two cell models is shown in Figure 5.2C. Both cell models can be expressed as pure NeuroML.

While these are quite simple models they were useful for comparing the behaviour of single compartment NeuroML based models across simulators. Figure 5.2D shows the time of the final spike due to the depolarising current in Figure 5.2A when the model is run on NEURON, GENESIS and MOOSE. As the time step is decreased, the three simulators converge to the same spike time. This has been one of the key findings of this work, how important a small integration time step is for accurately reproducing cell firing behaviour across simulators (section 6.2.1).

The work to develop a granule cell model reproducing a wide range of experimentally measured properties for use in large scale models is continuing in the Silver Lab. A number of other granule cell models are in the process of conversion to NeuroML, including one from a detailed 3D granule cell layer model (Solinas et al, 2010). This model improves on previous versions (Nieus et al, 2006) by improving the dependence of channel transition rates on temperature. Also, a new model of the granule cell based on the 2 variable Adaptive Exponential Integrate and Fire model (Brette and Gerstner, 2005) is being developed to match experimentally recorded cell behaviour, and is an ideal candidate for conversion to NeuroML v2.0.

Golgi cell models

The Golgi cell is an inhibitory interneuron present in the cerebellar granule cell layer. As mentioned previously, it is spontaneously active and provides

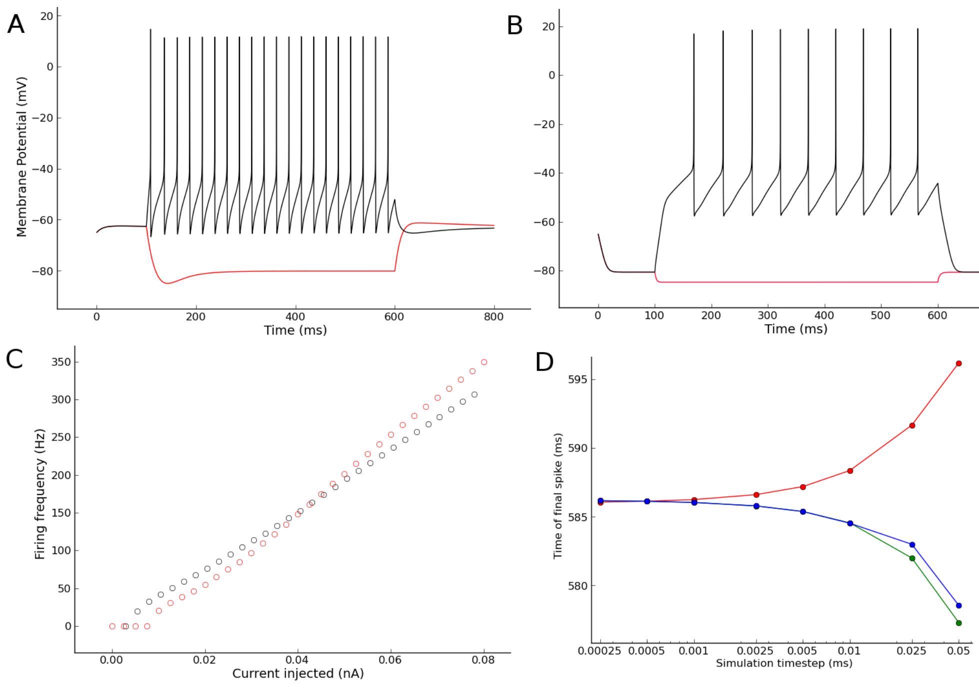


Figure 5.2: Cerebellar granule cell models. A) Model from Maex and Schutter (1998) showing response to depolarising (10pA, black trace) and hyperpolarising (-5pA, red trace) step currents. B) Model of Steuber and Saviane showing response to same step currents as A. C) Frequency versus input current responses of cells (black circles: cell from A, red: cell from B.) D) Time of last spike of cell from A as recorded on NEURON (red trace), GENESIS (green) and MOOSE (blue) as a function of the simulation timestep used.

feedback and feedforward inhibition to the granule cells. The network model of Maex and Schutter (1998) included a single compartment Golgi cell model which (due to lack of experimental data at the time) reused the channels of their granule cell model, introducing a further voltage shift in the activation rates to get the cell to fire spontaneously. Figure 5.3A shows the cell spiking without input and response to depolarising/hyperpolarising current clamp injections.

A multicompartmental Golgi cell model was produced in the lab of Egidio D'Angelo (Solinas et al, 2007a,b). This 5 compartment model (Figure 5.3B, inset) included 12 active conductances including, fast, resurgent and persistent Na^+ channels, delayed rectifier, A-type, slow repolarising M-like voltage gated K^+ channels, SK and BK [Ca^{2+}] dependent K^+ channels, high and low voltage activated Ca^{2+} channels, and 2 hyperpolarisation activated channels, HCN1, HCN2. Figure 5.3B shows spontaneous firing in the cell and responses to the same current clamp injections as Figure 5.3A.

The conductances of this cell model were reused by a model developed by Vervaeke et al (2010). This study reconstructed Golgi cell morphologies from which electrophysiological recordings had been made and incorporated these into detailed cell models. Only the passive properties of the cell models needed to be adjusted to reproduce the recorded firing properties of the cells. The cell structure and voltage trace under the same conditions as the other cells is shown in Figure 5.3C. This cell model formed the basis for an investigation of the properties of an electrically coupled Golgi cell network (section 5.1.4). Input current versus firing rate for all three Golgi cell models is shown in Figure 5.3C.

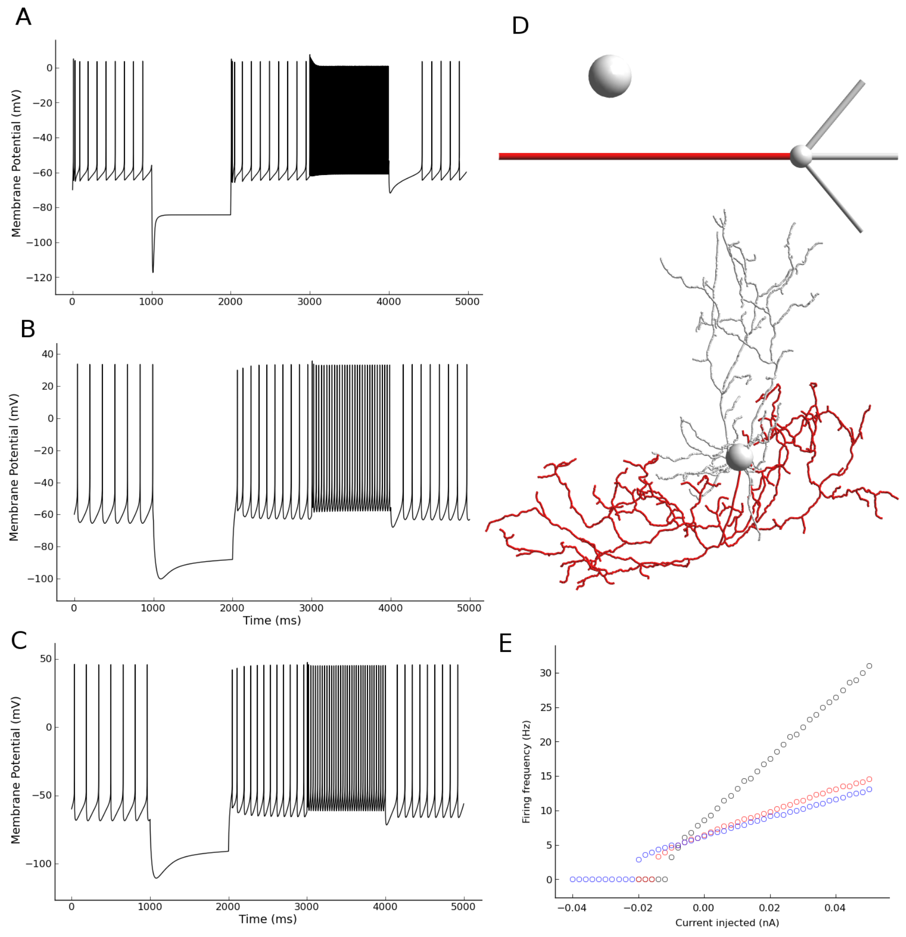


Figure 5.3: Golgi cell models. A) Single compartment model from Maex and Schutter (1998), showing response to hyperpolarising (-200pA for 1s at 1s) and depolarising (200pA for 1s at 3s) currents. Cell is spontaneously active with zero input current. B) Reduced model from Solinas et al (2007a) with responses to same currents as A. C) Detailed model from Vervaeke et al (2010) (4672 segments in 319 sections) with responses to same currents as A. D) 3D representations of the cells from A (top), B (middle, with part of axon truncated) and C (bottom) with axonal sections in red. E) Frequency versus input current responses of cells (black circles: cell from A, blue: cell from B, red: cell from C)

De Schutter and Bower (1994) Purkinje cell model

This model of the Purkinje Cell (De Schutter and Bower, 1994) was one of the first morphologically detailed cell models produced which attempted to incorporate the full range of active conductances and internal calcium dynamics present on a complex dendritic tree. The Purkinje cell is a prime target for this type of modelling due to its distinct planar structure, and the fact that that complex Ca^{2+} spikes can be triggered which spread throughout the dendritic tree. The model was originally constructed with GENESIS, and included fast and persistent Na^{+} channels, delayed rectifier, M-type and anomalous rectifier K^{+} channels, T- and P-type Ca^{2+} channels and two types of $[Ca^{2+}]$ dependent K^{+} channels.

The model was originally translated to NEURON by Arnd Roth using a morphology file generated by neuroConstruct, and then I converted the channels to NeuroML for a fully simulator independent version. Figure 5.4 shows the model running on NEURON and GENESIS, and the simulation visualised in neuroConstruct after recording the membrane potential in each segment during a simulation on NEURON. The conversion process highlighted a number of issues with the original implementation of the model including use of too few points in the precalculated tables of channel gating variable values (used to optimise the simulation). More appropriate values for these led to differences in spiking behaviour of the cell under constant current clamp (personal communication from Arnd Roth). This bug has persisted in a number of subsequent papers using this model. This highlights the need to test such popular models thoroughly and to keep records of any changes between versions.

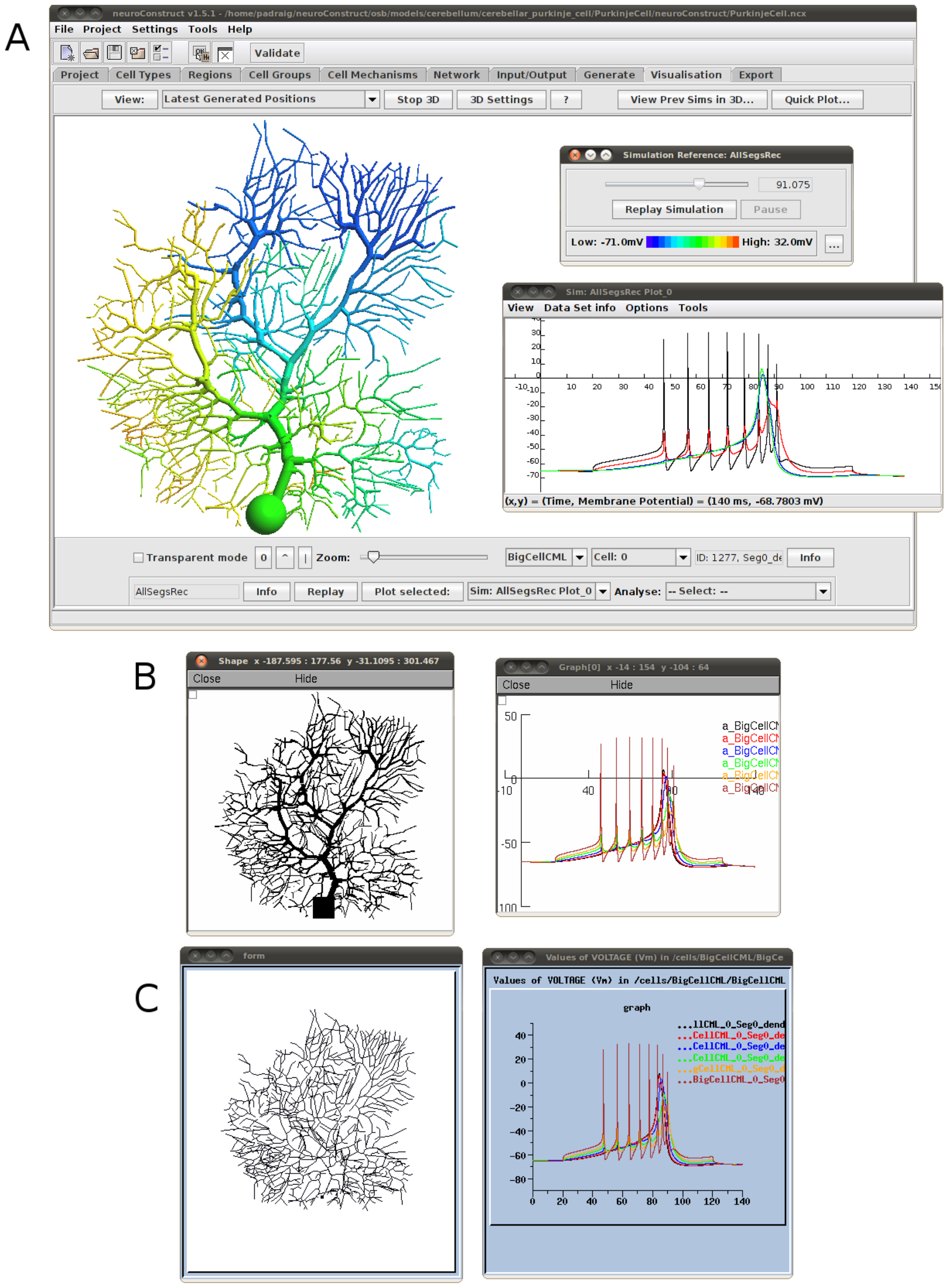

A
neuroConstruct v1.5.1 - /home/padraig/neuroConstruct/osb/models/cerebellum/cerebellar_purkinje_cell/PurkinjeCell/neuroConstruct/PurkinjeCell.ncx
File Project Settings Tools Help
Validate
Project Cell Types Regions Cell Groups Cell Mechanisms Network Input/Output Generate Visualisation Export
View: Latest Generated Positions Stop 3D 3D Settings ? View Prev Sims in 3D... Quick Plot...
Simulation Reference: AllSegsRec
91.075
Replay Simulation Pause
Low: -71.0mV High: 32.0mV
Sim: AllSegsRec Plot_0
View Data Set info Options Tools
(x,y) = (Time, Membrane Potential) = (140 ms, -68.7803 mV)
Transparent mode Zoom: BigCellCML Cell: 0 ID: 1277, Seg0_de Info
AllSegsRec Info Replay Plot selected: Sim: AllSegsRec Plot_0 Analyse: -- Select: --
B
Shape x -187.595 : 177.56 y -31.1095 : 301.467
Close Hide
Graph[0] x -14 : 154 y -104 : 64
Close Hide
a_BigCellCM
C
form
Values of VOLTAGE (Vm) in /cells/BigCellCML/BigCe
Values of VOLTAGE (Vm) in /cells/BigCellCML/BigCellCML
graph

Figure 5.4: De Schutter and Bower (1994) Purkinje cell model. A) Model visualised in neuroConstruct, including control panel for replaying simulation results, and plot of selected locations on cell (black trace for membrane potential at soma with red, blue, green increasingly distal points. B) Screenshot of neuroConstruct generated NEURON simulation. C) neuroConstruct generated GENESIS simulation.

5.1.4 Cerebellar network models

Maex and Schutter (1998) granule cell layer network model

This model was used to investigate the conditions which lead to synchrony of firing in the granule cell layer of the cerebellum (Maex and Schutter, 1998). The granule cell and Golgi cell models discussed above were used to construct a 1D model along the single parallel fibre axis, with synaptic properties and connectivity based on experimentally measured data. The study found that due to the reciprocal innervation between the granule cells and Golgi cells, the cells of the network quickly became entrained in a synchronous oscillation.

The 1D network was recreated in neuroConstruct to reproduce the basic network behaviour from the original study. To investigate the properties of the network in 3D, the cell models were extended by adding parallel fibre axons to to the granule cells (yellow cells in Figure 5.5A), and a single dendrite to the Golgi cells (green in Figure 5.5A). Excitatory connections were then made between the parallel fibres and Golgi cell dendrites (4 connections to dendrites close to the fibre) and inhibitory connections made to each granule cell from the nearest Golgi cell. Mossy fibres (blue in Figure 5.5A) were added and provided 50 Hz random synaptic input to the granule cells. Granule cells received between 3 and 7 (mean 4) of these inputs, selecting the closest mossy fibres to them. With these basic 3D properties the network was rerun and the synchrony investigated. Two beams within the network, aligned with the parallel fibres were identified (Figure 5.5B) and the synchrony of individual cells was compared to cells in the same beam and in the other. Cross-correlation of these cells revealed a higher correlation between cells in the same beam as compared to cells in separate beams. This behaviour is consistent with experimental results comparing simultaneous recordings from Golgi cell along and across parallel fibre tract (Vos et al, 1999).

This model demonstrates that network models can be generated and analysed in neuroConstruct with more realistic anatomical properties and behaviours than have been achieved previously. Note that most of the network construction, simulation and analysis described here can be carried out through the GUI of neuroConstruct without writing scripts.

A larger scale version of this network model was used for testing the parallel network generation of neuroConstruct (Figure 4.3).

Electrically connected Golgi cell network model

An investigation combining electrophysiology, immunohistochemistry, electron microscopy and detailed compartmental modelling was carried out to look at the behaviour of electrically coupled Golgi cell networks in the granule cell layer (Vervaeke et al, 2010). Koen Vervaeke in the Silver Lab carried out the majority of the work and I was involved in implementing the cell and network models in neuroConstruct, and adding new functionality to the application.

Paired electrophysiological recordings were made between Golgi cells to determine their electrical coupling. In addition to electrical recordings, a number of cells were filled with biocitin and their dendritic and axonal arbourisations reconstructed. Electron microscopy was used to locate the gap junctions between pairs of cells. Figure 5.6A shows a pair of reconstructed cells which have been loaded into neuroConstruct and the yellow spheres indicate the locations of the gap junctions which were discovered in this way. Immunoflourescent labelling of Connexin 36 and paired recordings in

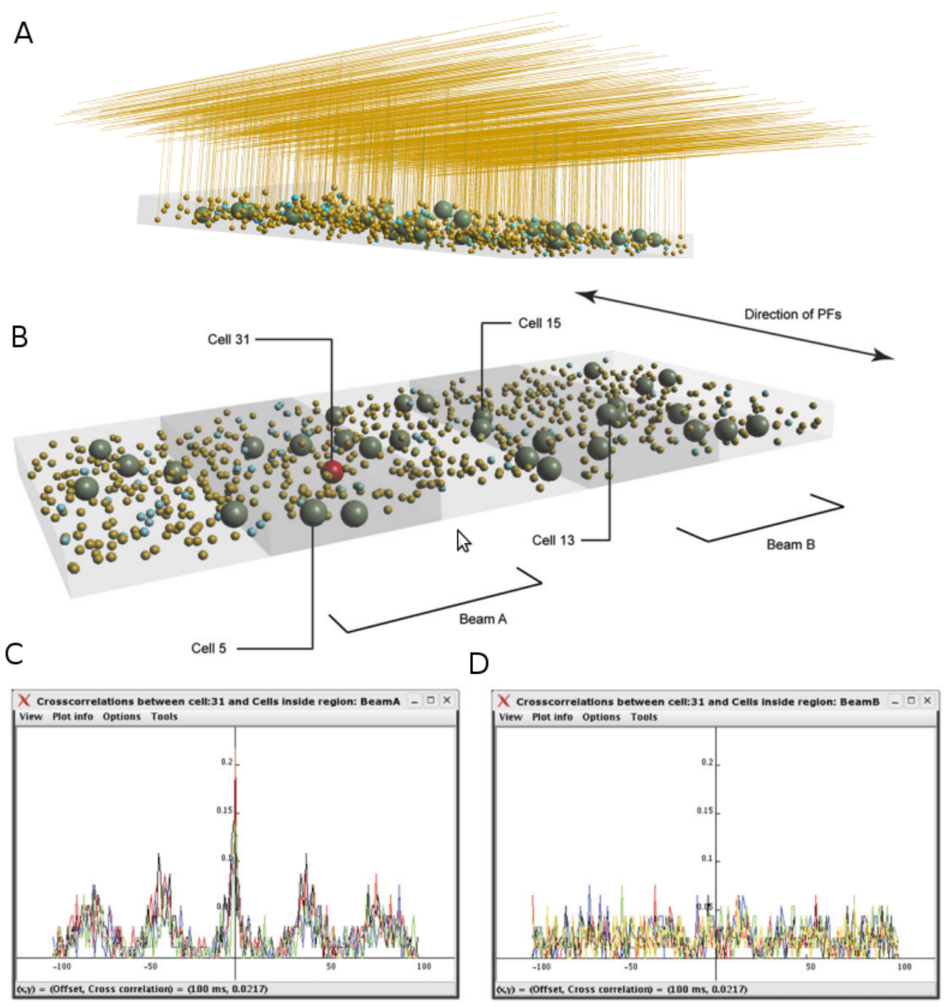

A
B
Cell 31
Cell 15
Direction of PFs
Cell 13
Beam B
Beam A
Cell 5
C
D
Crosscorrelations between cell:31 and Cells inside region: BeamA
View Plot info Options Tools
(x,y) = (Offset, Cross correlation) = (100 ms, 0.0217)
Crosscorrelations between cell:31 and Cells inside region: BeamB
View Plot info Options Tools
(x,y) = (Offset, Cross correlation) = (100 ms, 0.0217)

Figure 5.5: 3D granule cell layer model. A) Visualization of network model based on a published 1D model (Maex and Schutter, 1998). Mossy fibre terminals (blue), granule cell somata (orange), and Golgi cell somata (green) are packed in a 3D region (500μm in parallel fibre direction, 1mm parasagittally, 50μm in thickness) representing a section of the granule cell layer of the cerebellar cortex. The ascending segments and parallel fibres of the granule cells extend into the molecular layer region, as do the single dendrites of the Golgi cells. B) View of 3D cerebellar granule cell layer model showing only the cell bodies. Two regions are identified, beam A and beam B, which have nonoverlapping sets of parallel fibres. C) Cross-correlation between cell 31 and the other four GoCs in beam A, each color graph representing a different cell. The y axis represents the probability of a spike occurring in the other cell with the specified offset (1 ms time window). D) Crosscorrelation between cell 31 and the six GoCs in beam B, with identical axes to D. Modified from Gleeson et al (2007).

knockout mice were also used to show that that this protein is required for functional gap junctions between Golgi cells.

The effect of sparse, synchronous synaptic input on pairs of coupled cells was also investigated. It was observed that such sparse input could trigger antiphasic behaviour in electrically coupled cells which were otherwise firing synchronously (Figure 5.6B). This effect was due to the preferred transmission of the large afterhyperpolarization (AHP) as opposed to the brief spike between electrically coupled cells. This meant that the coupling between Golgi cells was chiefly inhibitory in nature.

Compartmental models were created using the reconstructed cells and we reused the set of conductances from a previous reduced Golgi cell model (Solinas et al (2007a), section 5.1.3). The passive properties of the cell models were adjusted to match the experimentally recorded the input resistance and gap junction conductance adjusted to match the coupling coefficient between the cells. The antiphasic firing of coupled cells following sparse input could also be reproduced in the model cells (Figure 5.6C).

The probability of a given Golgi pair being connected (Figure 5.7A) and the coupling strength (Figure 5.7B) was measured as a function of the inter soma distance. Data on the density of Golgi cells in the granule cell layer were obtained from our colleagues in Hungary (4607 ± 166 cells/mm^3).

All of these data were used to construct a realistic 3D model of an electrically coupled Golgi cell network in neuroConstruct (Figure 5.7C, D). Instances of one of the detailed Golgi cell models were randomly placed in a volume representing the granule cell layer at a realistic density (45 instances in a 350 x 350 x 80 μm volume, Figure 5.7D). These were then connected with gap junctions (a single simulated instance per pair to represent all individual gap junctions) and experimental data on coupling strengths and probabil-

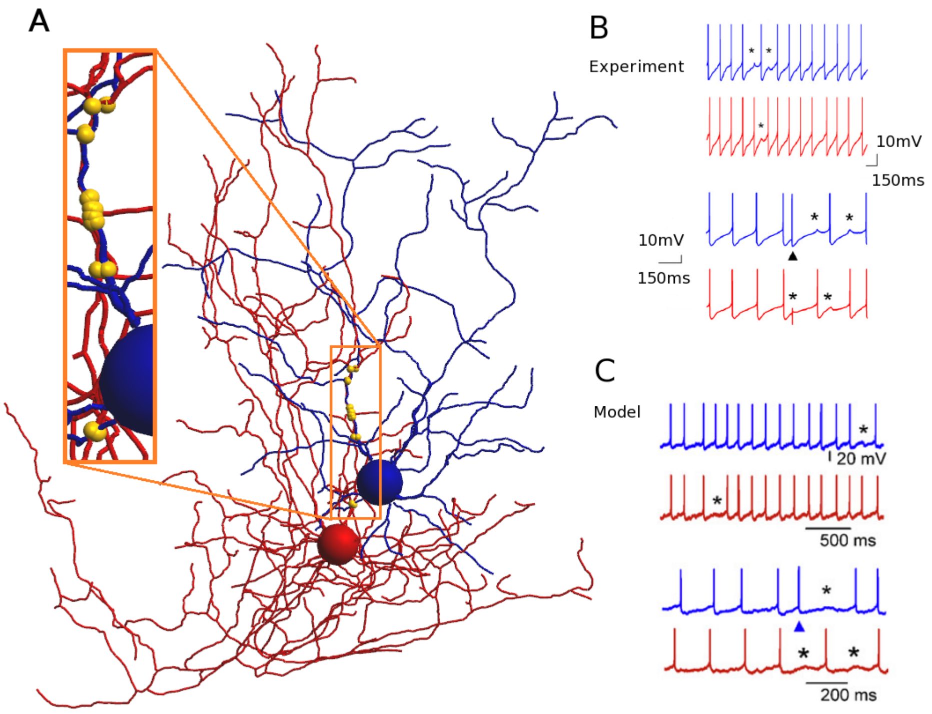


Figure 5.6: Pair of electrically connected Golgi cells from Vervaeke et al (2010). A) Two Golgi cells whose morphologies were reconstructed using Neurolucida, visualised in neuroConstruct after being imported into that application. The locations of gap junctions had been determined by electron microscopy analysis and are indicated by yellow spheres on each cell. B) Paired recording from a typical Golgi cell pair (not reconstructed pair) showing synchronous spiking (top), with asterisks indicating occasional skipped cycle leading to transient antiphase firing. Bottom pair of traces show response to out of phase synaptic stimulation (arrowhead). After this, the cells mutually inhibit one another through propagation of the AHP. C) Model behaviour of electrically connected cells from A receiving background noise designed to reproduce *in vitro* interspike interval variability. With no synaptic stimulation (top) synchronised spiking with occasional missed spikes (as in the experimental case) is observed. Bottom: on receipt of out of phase mossy fibre synaptic input (arrowhead), transient antiphase firing occurs. Modified from Vervaeke et al (2010).

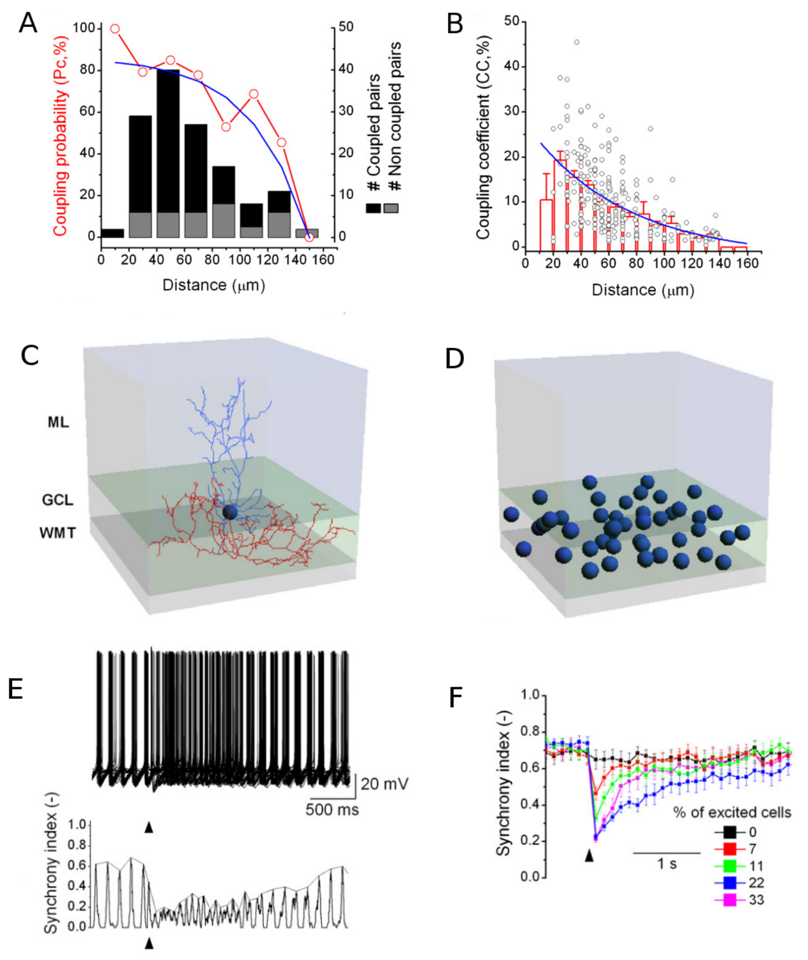


Figure 5.7: Golgi cell network model. A) Experimentally measured coupling probability between pairs of Golgi cells as a function of distance (136 pairs). Blue trace is fitted Bolzmann curve as used in neuroConstruct model. B) Coupling coefficients as a function of distance between somata (174 pairs). Histogram shows binning of data in 10μm bins. Blue trace is fitting with exponential decay function for use in neuroConstruct. C) 3D volume used for network model showing white matter tract (WMT), granule cell layer (GCL, 350 x 350 x 80 μm), molecular layer (ML) and a single cell as used in the network (soma and dendrites in blue, axon in red). D) Locations of 45 randomly placed Golgi cell somata. Neurites omitted for clarity. E) Membrane potential from all 45 cells in network model. Sparse synaptic stimulation was provided to 10 cells of the network (arrowhead). Before this the cells displayed loose synchrony, which disappeared on stimulation, but slowly returned. The synchrony index SI(t) is the total number of spikes within t and t + 20 ms normalized by the number of cells. F) Dependence of network desynchronisation on percentage of cells stimulated. Modified from Vervaeke et al (2010).

ity of connection were used to assign weights to each of these connections, dependent on distances between their somata. neuroConstruct was used to generate simulations of the networks for Parallel NEURON and run them remotely on UCL's Legion supercomputer (Table 4.1). The behaviour of the network when sparse excitatory synaptic stimulation is applied is shown in Figure 5.7E. The network, which had been displaying loose synchrony, became desynchronised before returning slowly to synchrony. The dependence of the network desynchronisation on the percentage of cells stimulated is shown in Figure 5.7F.

This investigation illustrated important aspects of the behaviour of electrically coupled networks under synchronous external stimulation. While electrical coupling is normally associated with greater synchrony between cells, in this case network desynchronisation was observed when a percentage of the cells in a coupled network received synchronous suprathreshold input. This result is important when considering the role of gap junctions in the brain, and specifically for information processing in the granule cell layer, where the Golgi cell network will have a different influence in its synchronised and desynchronised states.

The use of neuroConstruct in this investigation was useful not only for generating the 3D connectivity, but also in its ability to generate and manage multiple instances of the networks, both with different random structure and different stochastic inputs. Generating spatially heterogeneous synaptic connectivity was also crucial, as the effect demonstrated was highly dependent on network heterogeneity.

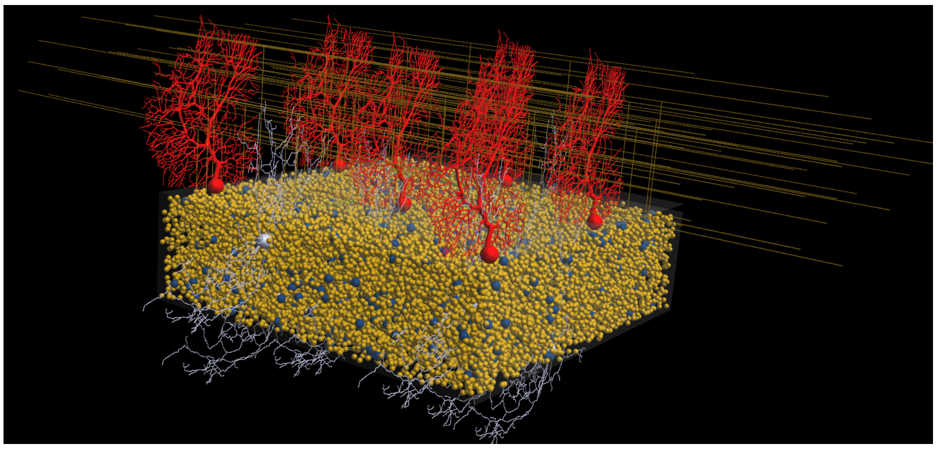


Figure 5.8: Initial version of 3D model of cerebellar cortex incorporating mossy fibre input, granule cells with parallel fibres and detailed Purkinje cell and Golgi cell models.

Towards large scale 3D cerebellar network models

In addition to the cerebellar cell models discussed previously, other detailed conductance based models are in the process of conversion to NeuroML. These include a detailed Deep Cerebellar Nucleus (DCN) cell model (Steuber et al, 2011), and a single compartment stellate cell model (Molineux et al, 2005). All of these cell models will be incorporated into a large scale network model of the whole cerebellar circuitry. An initial version of the model is shown in Figure 5.8.

Planning for this is only at an early stage, and it will take coordinated work with other groups interested in the various cellular elements and signalling pathways of the cerebellar circuitry to assemble a full model. Chapter 7 discusses initiatives which are in development for enabling this type of collaborative, large scale model development to take place.

5.2 Neocortical model development

5.2.1 Anatomical and functional connectivity of the neocortex

The cerebral cortex is the region of the brain which is responsible in large part for the complex information processing power of the mammalian brain. Functions such as memory, attention, decision making and in humans, language and conciousness are all associated with this structure. Its largest part, the neocortex, is organised into six layers, containing an array of excitatory pyramidal cells making dense local and long range synaptic connections, and a variety of local interneurons (Shepherd, 2004). Each layer has cells with distinct firing properties, morphological characteristics and preferred innervation domains (Thomson and Lamy, 2007). These cells are organised in vertically oriented functional columns which span all layers and act as the basic processing units of the cortex. A key aspect of understanding how sensory signals are processed by the cortex will be to elucidate the underlying neuronal microcircuit in these columns, the pattern of connectivity between cell types which is repeated throughout most of the cortex and across species (Douglas and Martin, 2007).

Towards this goal, many researchers have chosen to concentrate on specific brain regions, associated with identifiable sensory processing or motor output. The visual cortex is a very popular region for investigating the functional circuits underlying processing of sensory information with new experimental techniques (Ko et al, 2011; O'Connor et al, 2009). The rodent barrel cortex has also received a lot of attention due to the close correspondence of the sensory vibrissal input to the anatomy. Input from each whisker projects via the thalamus to an identifiable "barreloid" region in the contralateral somatosensory cortex (Petersen, 2007). The columnar regions are most readily identifiable in layer 4 in stained tissue. Much experimental work has been done mapping connectivity and signal flow in this region (Lefort et al, 2009; Lübke and Feldmeyer, 2007; Lübke et al, 2003; Silver et al, 2003). New experimental techniques are constantly being developed for probing the circuitry of the cortex (O'Connor et al, 2009; Rancz et al, 2011) and other brain regions. In addition to the promise of large scale reconstruction of neuronal connectivity through serial electron microscopy techniques (Briggman and Denk, 2006) and noninvasive functional dissection of circuitry using optogenetic techniques (Boyden et al, 2005), advances in microscope technology promise new experimental data on simultaneous activity at multiple cellular locations with much greater temporal resolution both *in vitro* and *in vivo* (e.g. acousto-optic lensing microscope being developed in Silver Lab, Kirkby et al (2010)).

5.2.2 Neocortical modelling

The cortex has always been of great interest to theoreticians and computational modellers. Models of information processing in cortical networks range from very abstract models of interacting cortical regions in cognitive models down to models including individual spiking neurons and anatomically based connectivity patterns. Neuronal elements in these can be represented in varying levels of detail (Figure 1.1B), and modelling studies are published at all of these levels from simple abstract cell models (Destexhe, 2009; Kumar et al, 2008; Lefort et al, 2009; Vogels and Abbott, 2005), minimal multicompartmental cells (Djurfeldt et al, 2008; Izhikevich and Edelman, 2008) to quite detailed cell models incorporating multiple cellular regions and realistic distributions of membrane conductances for a range of cortical and thalamic cell types (Traub et al, 2005). Initiatives have been started to create models on the scale of cortical columns with even greater levels of detail, where all cells have realistic morphologies and all known synaptic connectivity is included (Markram, 2006), and this is also a goal of researchers interested in information processing in the barrel cortex (Helmstaedter et al, 2007). The utility of these initiatives will not only be in providing detailed computational resources for helping to explain experimental findings and guide new research directions, but will also be in the consolidation and sharing of of knowledge of these complex circuits between researchers.

The models I have focussed on for conversion to neuroConstruct and NeuroML include detailed pyramidal cell models and the network models of Roger Traub and colleagues containing cells with reduced numbers of compartments.

5.2.3 Neocortical and thalamic cell models

Mainen et al (1995) Layer 5 Pyramidal cell

This was one for the first published models (Mainen et al, 1995) to use a very detailed neuronal reconstruction (in this case a rat layer 5 pyramidal cell) to investigate propagation of action potentials in dendrites with active conductances. It was inspired by earlier experimental work (Stuart and Sakmann, 1994) investigating the backpropagation of action potentials into dendritic trees, and was also used to investigate the biophysical underpinnings of action potential initiation in the axon initial segment (AIS).

The model itself is quite simple, with just one fast Na^+ and one delayed rectifier K^+ conductance, but this is sufficient to demonstrate how action potentials initiate at the AIS, independently of where a steady current is injected into the cell, and propagate back through the dendritic tree (Figure 5.9).

Pyramidal cell model from Kole et al (2008) and Rothman et al (2009)

This cell model, also of a layer 5 pyramidal cell, was used by Kole et al (2008) in conjunction with antibody staining, whole cell patch clamping and Na^+ imaging in a more detailed study of the location for initiation of the action potential. They found that a higher density of Na^+ channels at the AIS was indeed crucial for action potential initiation at that point, and that this also accounted for the experimentally observed rate of rise of axonal action potential. The model itself was much more detailed than that of Mainen et al (1995), using in addition to the reconstructed dendritic tree, a manually constructed axon with initial segment and myelinated sections between nodes of Ranvier. The channel complement comprised Na^+ channels, high voltage activated, low voltage activated (Kv1 like) and slow, non inactivating M-type K^+ channels, and I_h channels.

This cell was reused in the study by members of the Silver Lab into the effects of synaptic depression on gain modulation in simple and complex cells (Rothman et al, 2009). The study used both dynamic clamp experiments on granule cells in acute cerebellar slices and computational modelling to investigate the changes in gain modulation in the presence and absence of short term depression (STD). Experimental work and modelling of simple cells showed that synaptic depression could change an additive shift in gain into multiplicative gain modulation. The Kole et al (2008) pyramidal cell model was used to study this effect in more complex cells. Excitatory and inhibitory synaptic inputs were targeted at the basal dendrites (Figure 5.10A)

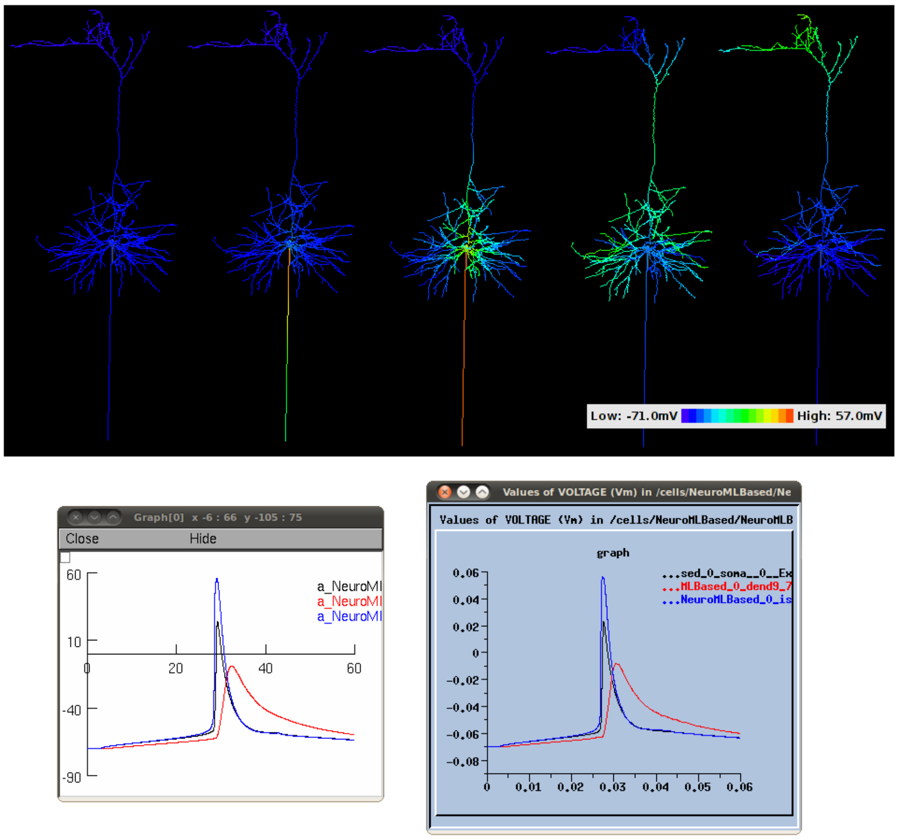


Figure 5.9: Mainen et al (1995) layer 5 pyramidal cell model. Top: initiation of action potential at axon initial segment and propagation back into the dendritic tree. Images taken from neuroConstruct replay of cell activity at 5 time points. Bottom: screenshots of plots from NEURON (left) and GENESIS (right) showing membrane potential at the axon initial segment (blue), soma (black) and a point 416μm along the apical dendrite (red).

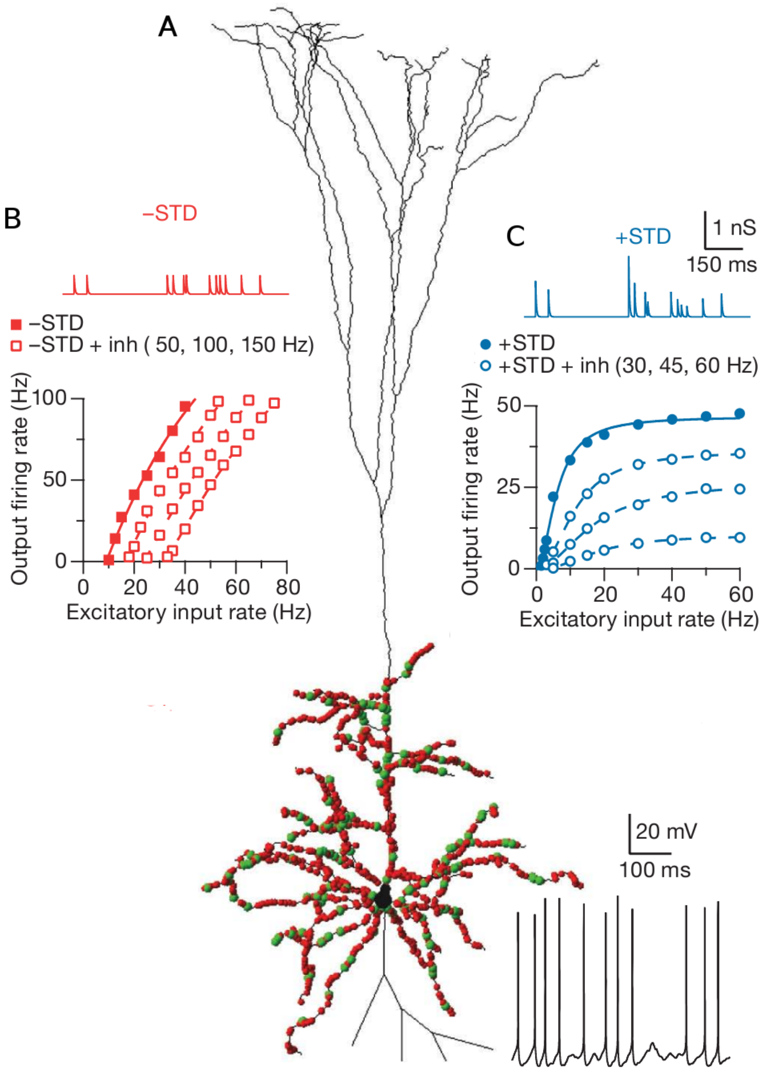


Figure 5.10: Multiplicative gain modulation in a layer 5 pyramidal cell model. A) Multicompartmental neuron model of layer 5 pyramidal cell based on Kole et al (2008) showing locations of excitatory (red circles) and inhibitory (green) synaptic inputs. Trace in lower right shows typical spiking behaviour during excitation and inhibition. B) Top: conductance train for a single excitatory synapse without STP. Bottom: input output relationship for cell under non depressing excitation with no inhibition (solid squared) and varying levels of inhibition (open squares). C) Conductance train (top) and input output relationship as in B, but with STP at the excitatory synapse. Modified from Rothman et al (2009).

using neuroConstruct, and the effect of increased inhibitory drive on the gain of the cell was investigated with (Figure 5.10B) and without (Figure 5.10C) STD at the excitatory synapses. In this case also, depression at the inhibitory synapses could change a largely additive effect on excitatory gain into a multiplicative operation.

The use of neuroConstruct in this study facilitated cell region specific placement of the synaptic connections, generation of multiple simulations, and management and analysis of data, as the individual simulations were stored in structured folders with metadata on simulation parameters.

Traub et al (2005) thalamocortical cell models

The thalamocortical network model of Traub et al (2005) is one of the most complex multi-cellular network models published to date. It is the culmination of over 20 years of modelling work, mainly by Roger Traub and contains models of the principle cell types from multiple layer of the cortex and the thalamus (Table 5.2) with semi realistic morphologies and realistic membrane conductances, and incorporates detailed connectivity with excitatory, inhibitory and electrical synaptic contacts between the cell types. The sheer complexity and level of detail incorporated can lead to questions on how useful such a model can be (see commentary on model by Kopell (2005)). The authors freely admit this, and are at pains to point out the preliminary and incomplete nature of the model. Nevertheless it is very valuable asset as a painstakingly assembled amalgamation of what is known about the conductances underlying spiking in these cell types and the synaptic connectivity properties between each cell type.

Such a consolidation of data makes it explicit whether a certain channel or connection type has been taken into account. If it is present, questions

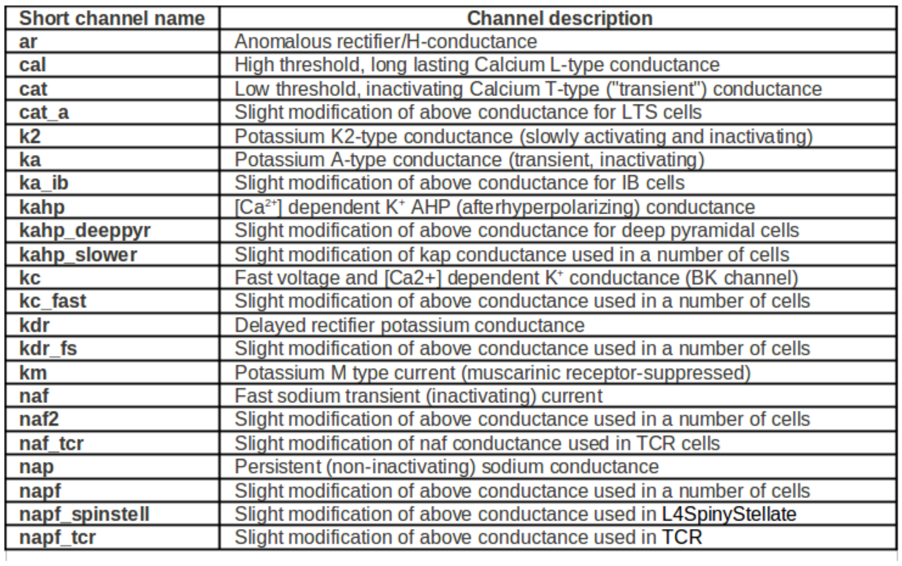
Short cell name	Cell description
L23PyrRS	Layer 2/3 Regular Spiking pyramidal cell
L23PyrFRB	Layer 2/3 Fast Rhythmically Bursting pyramidal cell
SupBask	Superficial Basket cell
SupAxAx	Superficial Axo-axonic cell
SupLTS	Superficial Low Threshold spiking cell
L4SpinStell	Layer 4 Spiny Stellate cell
L5TuftIB	Layer 5 Tufted Intrinsically Bursting pyramidal cell
L5TuftRS	Layer 5 Tufted Regular Spiking pyramidal cell
L6NonTuftRS	Layer 6 Non-tufted Regular Spiking pyramidal cell
DeepBask	Deep Basket cell
DeepAxAx	Deep Axo-axonic cell
DeepLTS	Deep Low Threshold spiking cell
TCR	Thalamocortical relay cell
nRT	Nucleus reticularis thalami cell

Table 5.2: List of cell types used in Traub et al (2005) model. Note that SupBask, SupAxAx, DeepBask and DeepAxAx had identical morphologies and sets of conductances, but were distinguished in the network model by their connectivity. SupLTS and DeepLTS were similarly electrically equivalent, meaning there were just 10 electrophysiologically distinct cells.

can be asked about the significance of the feature for the overall network behaviour. The model can also provide a valuable resource for theoreticians who want to use some element of biological detail in more abstract models. By process of comparison of detailed models with simplified ones, questions can be addressed about what features of the system are essential to a specific signal processing task.

A drawback of the original model was that it was developed in custom FORTRAN code, over 10,000 lines of code hard coded to run on 14 compute nodes (one cell type per node). The model was initially converted to NEURON by Michael Hines and Tom Morse, and I, together with Yoana Dimitrova and Simon Barnes, rotation students in the Silver Lab, converted the cell models to NeuroML and recreated the network connectivity in neuroConstruct.

The electrical behaviour of the model arises from 22 voltage- and ligand-gated Na^+ , K^+ and Ca^{2+} conductances together with both electrical and chemical synapses, which were all converted to ChannelML and tested (Table 5.3). I checked that these channels displayed identical behaviour across NEURON, GENESIS and MOOSE when incorporated in a single compartment cell for testing (Figure 5.11A). Each of the 14 cell types present was converted to NeuroML, using the Level 2 cell export function of NEURON (section 3.5.5) and import function of neuroConstruct (section 3.5.4). The different complements of the channels and different morphologies gave rise to a variety of behaviours including regular spiking, fast spiking and bursting behaviour (Figure 5.11B-E). The NeuroML implementation produced qualitatively similar spiking behaviour for simulations run in the 3 simulators in the 10 electrophysiologically distinct cells during sustained firing over hundreds of milliseconds to seconds. However, differences in the timing of spikes was evident in some of the cells, unless the spatial and temporal discretisation of the cell was increased substantially.



Short channel name	**Channel description**
ar	Anomalous rectifier/H-conductance
cal	High threshold, long lasting Calcium L-type conductance
cat	Low threshold, inactivating Calcium T-type ("transient") conductance
cat_a	Slight modification of above conductance for LTS cells
k2	Potassium K2-type conductance (slowly activating and inactivating)
ka	Potassium A-type conductance (transient, inactivating)
ka_ib	Slight modification of above conductance for IB cells
kahp	$[Ca^{2+}]$ dependent K^{+} AHP (afterhyperpolarizing) conductance
kahp_deeppyr	Slight modification of above conductance for deep pyramidal cells
kahp_slower	Slight modification of kap conductance used in a number of cells
kc	Fast voltage and [Ca2+] dependent K^{+} conductance (BK channel)
kc_fast	Slight modification of above conductance used in a number of cells
kdr	Delayed rectifier potassium conductance
kdr_fs	Slight modification of above conductance used in a number of cells
km	Potassium M type current (muscarinic receptor-suppressed)
naf	Fast sodium transient (inactivating) current
naf2	Slight modification of above conductance used in a number of cells
naf_tcr	Slight modification of naf conductance used in TCR cells
nap	Persistent (non-inactivating) sodium conductance
napf	Slight modification of above conductance used in a number of cells
napf_spinstell	Slight modification of above conductance used in L4SpinyStellate
napf_tcr	Slight modification of above conductance used in TCR

Table 5.3: List of active conductances used in Traub et al (2005) model.

Two observations confirmed that the main cause of divergence in spike times arose from the use of symmetrical compartments (where axial resis-

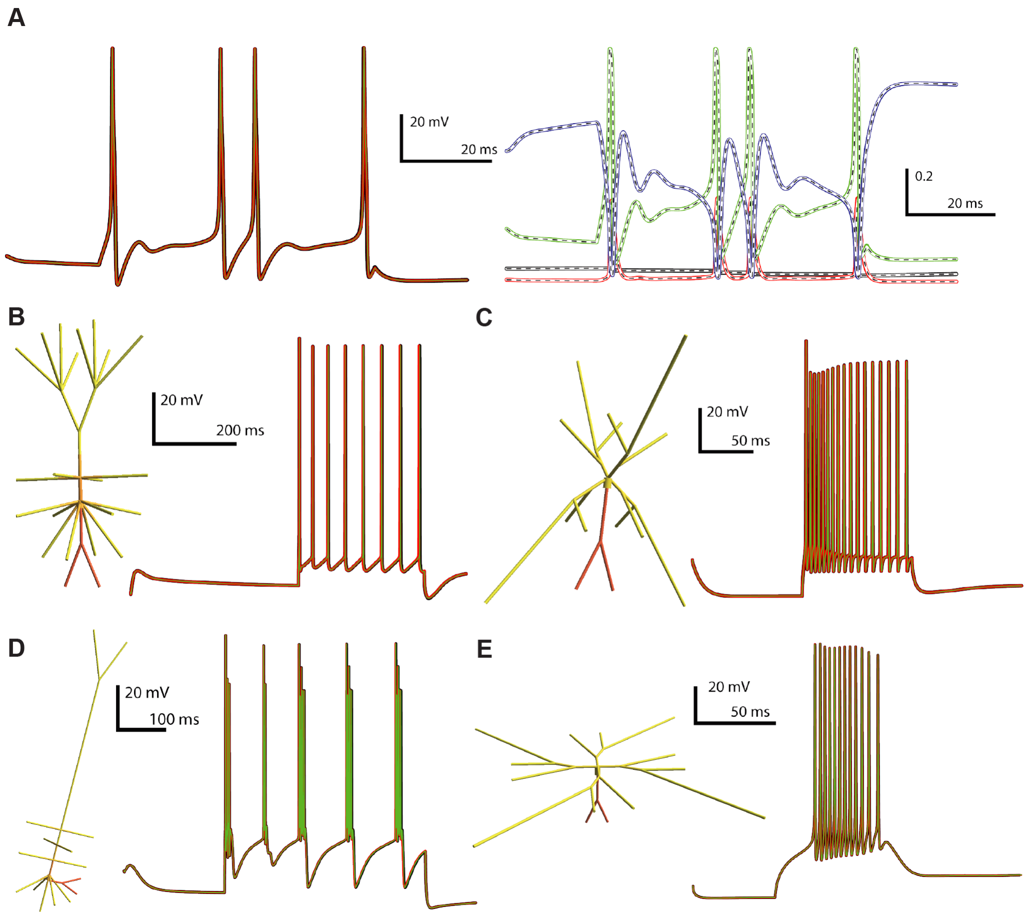


Figure 5.11: A number of cell models from Traub et al (2005). A) Single compartment cell model containing all 22 active conductances present in the detailed cell models, together with a passive conductance and a decaying calcium pool. Left plot shows the membrane potential response to a 80 pA current injection on NEURON (black), GENESIS (red) and MOOSE (green). Simulation timestep in 0.025 ms. Right plot shows the behaviour on NEURON of the activation variables for the anomalous rectifier (thick black line), L-type Ca^{2+} (red) and persistent Na^+ conductances (green) and the inactivation variable of the fast Na+ conductance (blue). White curve overlays show the corresponding GENESIS traces, and dashed lines show MOOSE traces. B-E) 3D representations of four cell models from Traub et al (2005) implemented in NeuroML, color indicates the density of fast sodium conductances on the cell membrane (red: high - yellow: low). Graphs show somatic membrane potential during current injections for: B) regular spiking (RS) Layer 2/3 pyramidal cell; C) superficial low threshold spiking (LTS) interneuron; D) intrinsically bursting (IB) Layer 5 pyramidal cell; E) nucleus reticularis thalami (nRT) cell (trace colors as for left panel of A).

tance is split and numerical integration takes place at the center of the compartment) and asymmetrical compartments (axial resistance is located at one end of the compartment). Firstly, the spike times of a single compartment cell with all the channel conductances included were indistinguishable on NEURON, GENESIS and MOOSE (Figure 5.11A), confirming the ChannelML implementations allowed equivalent behaviour on all 3 simulators. Secondly, when the spatial discretisation of the cell models was increased, all simulators tended toward the same spike times (Figure 5.12), with GENESIS generally requiring a finer discretisation (see section 6.2.1 for a discussion on the cause of this). These results show that the way models are implemented on different simulators can have a significant impact on their behaviour. Moreover, true interoperability, as measured through model convergence, may only occur at the limits of spatial and temporal discretisation (section 6.2.1).

5.2.4 Thalamocortical network models

Once all the channel, synaptic and cellular components of the Traub et al (2005) model were converted to NeuroML and tested, I used neuroConstruct to build a 56 cell Layer 2/3 network that matched as closely as possible a network model which uses a number of these cells (Cunningham et al, 2004). This consisted of regular spiking and fast rhythmic bursting pyramidal cells and low threshold spiking, axo-axonic and basket type interneurons (Figure 5.13A). As specified in the original model, excitatory and inhibitory synaptic conductances were located on specific dendritic and somatic segments and electrical synapses were included within cell populations. This network model was not tuned against any new experimental data and was primarily intended as a test case for comparison of network behaviour across simula-

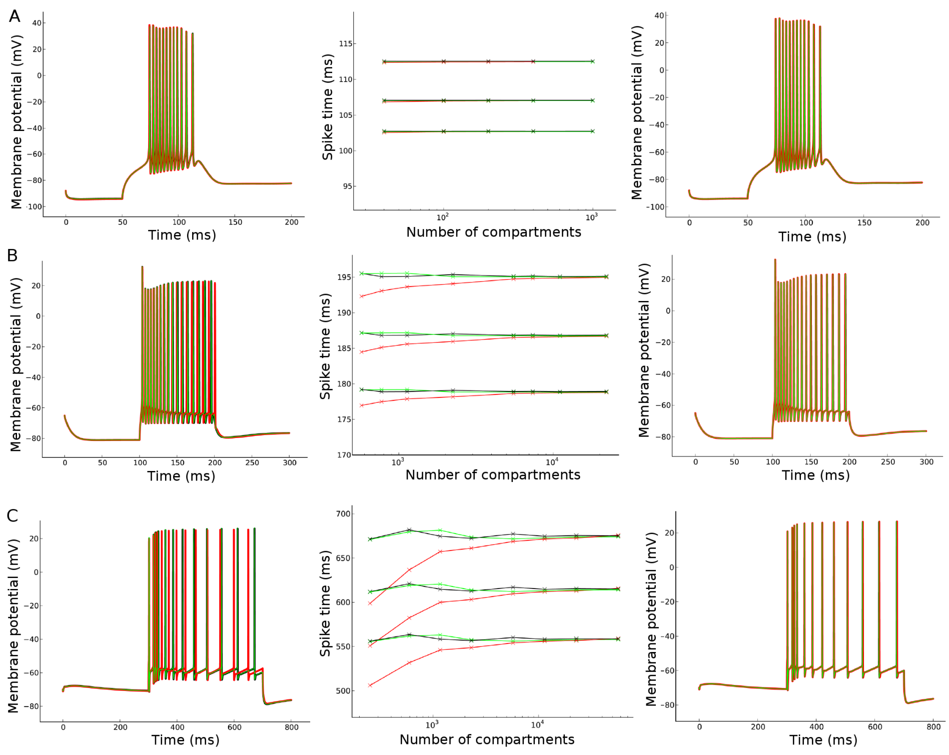


Figure 5.12: Comparison of cell models under NEURON (black), GENESIS (red) and MOOSE (green) as spatial discretisation is made finer. A) Nucleus reticularis thalami (nRT) cell run with total number of numerical simulation points (total nseg) in NEURON between 446 (left) and 4222 (right). Note that the number of compartments in GENESIS/MOOSE corresponds roughly to this figure, the spatial recompartmentalisation algorithm discussed in section 2.2.1 was used. Center plot shows the times of the last 3 spikes. B) Superficial Low Threshold spiking (LTS) cell with plots as in A. The left traces are for a cell with 573 numerical simulation points, the right is for 22094. C) Layer 6 Non-tufted Regular Spiking pyramidal cell with plots as in A. The left traces are for a cell with 252 numerical simulation points, the right is for 56976. Note that the increased spatial discretisation is applied uniformly across the cell based on the passive properties. Better algorithms for deciding spatial discretisation on each region of the cell based on local densities of active conductances (possibly customized for each simulator) should lead to convergence of spike traces with smaller number of numerical simulation points/compartments

tors. The spike times of the neuronal populations were similar across the 3 simulators over the first 200 ms of the simulation, when a small simulation timestep and fine spatial discretisation was used (Figure 5.13B). At longer times, some spikes became shifted and others appeared or disappeared depending on the simulator. This divergence in model behaviour occurred earlier in the simulation run and was much more pronounced when a more typical time step and coarser discretisation was used, suggesting that in practice, the precise spike times, and even the occurrence of some spikes produced by complex network models, will depend on the simulator implementation. A complete description of this network model including cell structure, channels, synapses, and lists of cell locations and connections can be represented in a single Level 3 NeuroML file.

The full connectivity of the Traub et al (2005) network model was then implemented. The network connection properties from Appendix B of that paper were incorporated into neuroConstruct. Figure 5.14 shows the connectivity matrix between the cell populations. As this is defined by giving the number of incoming synaptically connections on each cell in a population it is independent of network size. The figure was generated automatically by neuroConstruct. More detailed views are possible giving more details on the synaptic properties of each connection type.

Due to the size of the network, Parallel NEURON is the simulator most appropriate for executing this model. It will also run on GENESIS and MOOSE, but networks are limited to approximately 100 cells due to lack of parallel support for those simulators in neuroConstruct. While the network can be generated through the GUI, a Python script was developed which generates network instances scaled from the original 3,560 cell network. An instance of the network with 10% of the cells is shown in Figure 5.15A.

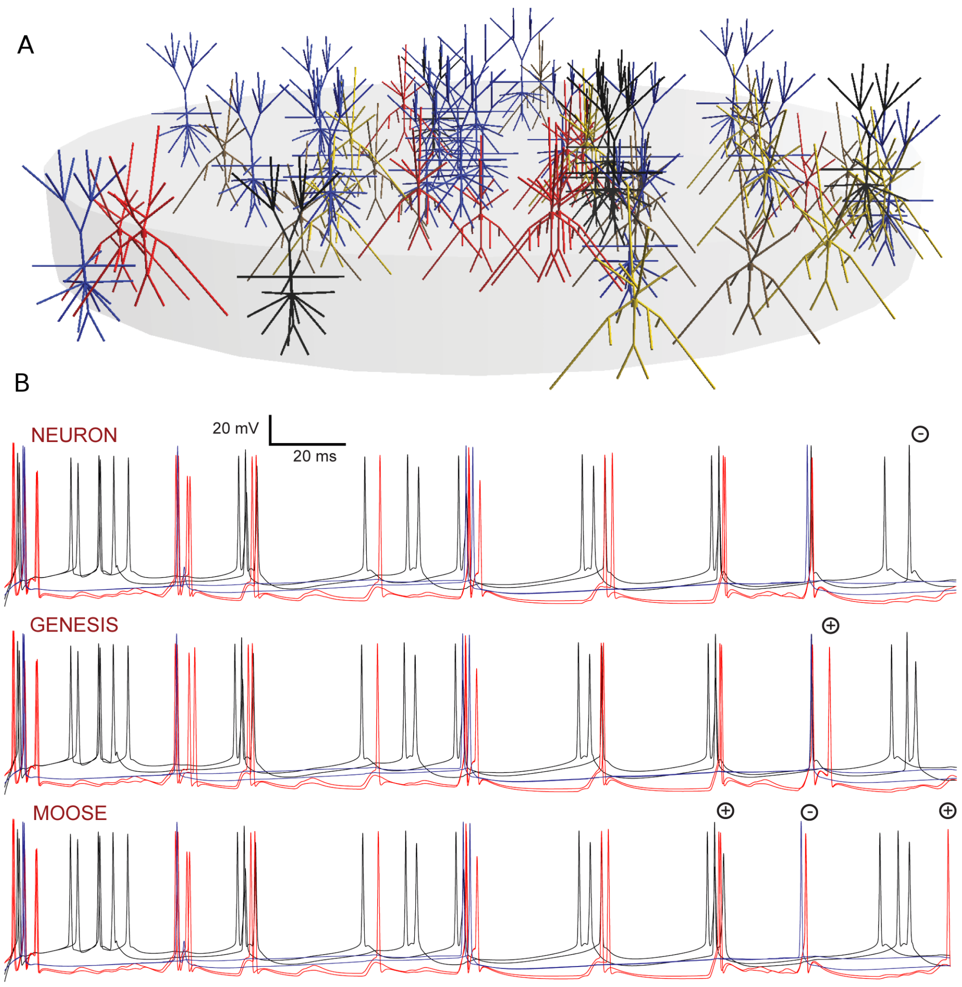
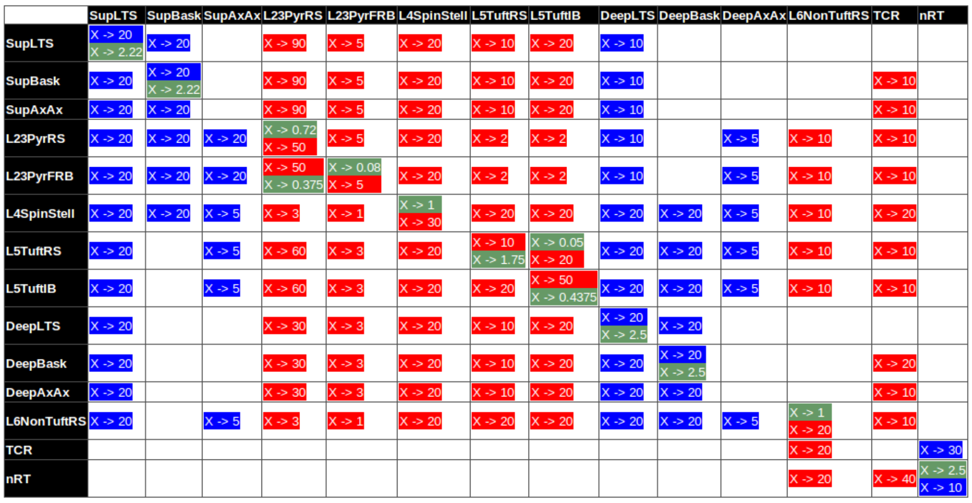


Figure 5.13: Comparison of the behaviour of a NeuroML-based Layer 2/3 network model with 5 cell types connected with both electrical and chemical synaptic connections run on NEURON, GENESIS and MOOSE simulators. The network is based on the larger network described in Cunningham et al (2004), and uses five of the cortical cell models converted to NeuroML from Traub et al (2005). A) 20 regular spiking pyramidal cells (RS, blue), 6 fast rhythmic bursting pyramidal cells (FRB, black), 10 low threshold spiking interneurons (LTS, red), 10 axo-axonic interneurons (yellow) and 10 basket cells (brown) placed at random in a cylindrical region. The network contained electrical connections between the cells within each population, along with 4300 excitatory connections of 10 types within and between populations and 3800 inhibitory connections of 12 types, but these are not shown. B) Somatic membrane potential traces from 2 each of RS, FRB and LTS cells (with colours as in A) for simulations run on NEURON (top), GENESIS (middle) and MOOSE (bottom). Simulation time step was 0.001 ms.



	SupLTS	SupBask	SupAxAx	L23PyrRS	L23PyrFRB	L4SpinStell	L5TuftRS	L5TuftIB	DeepLTS	DeepBask	DeepAxAx	L6NonTuftRS	TCR	nRT
SupLTS	X -> 20 X -> 2.22	X -> 20		X -> 90	X -> 5	X -> 20	X -> 10	X -> 20	X -> 10					
SupBask	X -> 20	X -> 20 X -> 2.22		X -> 90	X -> 5	X -> 20	X -> 10	X -> 20	X -> 10				X -> 10	
SupAxAx	X -> 20	X -> 20		X -> 90	X -> 5	X -> 20	X -> 10	X -> 20	X -> 10				X -> 10	
L23PyrRS	X -> 20	X -> 20	X -> 20	X -> 0.72 X -> 50	X -> 5	X -> 20	X -> 2	X -> 2	X -> 10		X -> 5	X -> 10	X -> 10	
L23PyrFRB	X -> 20	X -> 20	X -> 20	X -> 50 X -> 0.375	X -> 0.08 X -> 5	X -> 20	X -> 2	X -> 2	X -> 10		X -> 5	X -> 10	X -> 10	
L4SpinStell	X -> 20	X -> 20	X -> 5	X -> 3	X -> 1	X -> 1 X -> 30	X -> 20	X -> 20	X -> 20	X -> 20	X -> 5	X -> 10	X -> 20	
L5TuftRS	X -> 20		X -> 5	X -> 60	X -> 3	X -> 20	X -> 10 X -> 1.75	X -> 0.05 X -> 20	X -> 20	X -> 20	X -> 5	X -> 10	X -> 10	
L5TuftIB	X -> 20		X -> 5	X -> 60	X -> 3	X -> 20	X -> 20	X -> 50 X -> 0.4375	X -> 20	X -> 20	X -> 5	X -> 10	X -> 10	
DeepLTS	X -> 20			X -> 30	X -> 3	X -> 20	X -> 10	X -> 20	X -> 20 X -> 2.5	X -> 20				
DeepBask	X -> 20			X -> 30	X -> 3	X -> 20	X -> 10	X -> 20	X -> 20	X -> 20 X -> 2.5			X -> 20	
DeepAxAx	X -> 20			X -> 30	X -> 3	X -> 20	X -> 10	X -> 20	X -> 20	X -> 20			X -> 10	
L6NonTuftRS	X -> 20		X -> 5	X -> 3	X -> 1	X -> 20	X -> 20	X -> 20	X -> 20	X -> 20	X -> 5	X -> 1 X -> 20	X -> 10	
TCR												X -> 20		X -> 30
nRT												X -> 20	X -> 40	X -> 2.5 X -> 10

Figure 5.14: Connectivity matrix for Traub et al (2005) model. Presynaptic cell populations are listed along the top, postsynaptic along the left. Red indicates excitatory, blue inhibitory and green electrical connections. The numbers indicate how many inputs are received from presynaptic cells by each postsynaptic cells, e.g. the top right red box indicates that each superficial basket cell receives excitatory synaptic input from 10 TCR cells. This view of the connectivity can be automatically generated from the neuroConstruct project for creating the network.

While the cells are placed in 3D, the connectivity is essentially the same as the original 1D model. Cells target the preferred dendritic locations on postsynaptic cells, and these connections are made independently of the cells' positions. Traces of the typical behaviour of some of the cells in the network are shown in Figure 5.15B.

The development of this 3D extension of the model is in its early stages. It has proven very useful for demonstrating the scope of NeuroML, the cross simulator convergence of multicompartmental cell model behaviour, the ability of neuroConstruct to generate and visualise large complex 3D network models and has been useful as a test case for parallel network simulations (section 4.4). There are however a number of issues with the model which will need to be addressed, some to do with the neuroConstruct/NeuroML implementation of the Traub model, others to do with the physiological properties of the model.

The NMDA synapse model used in the original paper had a linear rise and exponential decay conductance waveform. This had to be approximated by the double exponential form supported in NeuroML. The cell models required a much higher spatial discretisation (on the order of hundreds of compartments) even in NEURON, compared to the ∼60-70 used by Traub and colleagues. Evidence suggests that the cell models used were not optimally spatially discretised, and the precise spike timing would be dependent on their numerical solver for the cells.

With regards to the behaviour of the cell models, some of them, notably the interneurons, were spontaneously active. All of the cell models would benefit from retuning against new experimental data, and ideally the data should be made available along with the model. Work is ongoing in the Silver Lab to develop new algorithms, closely tied with neuroConstruct, to

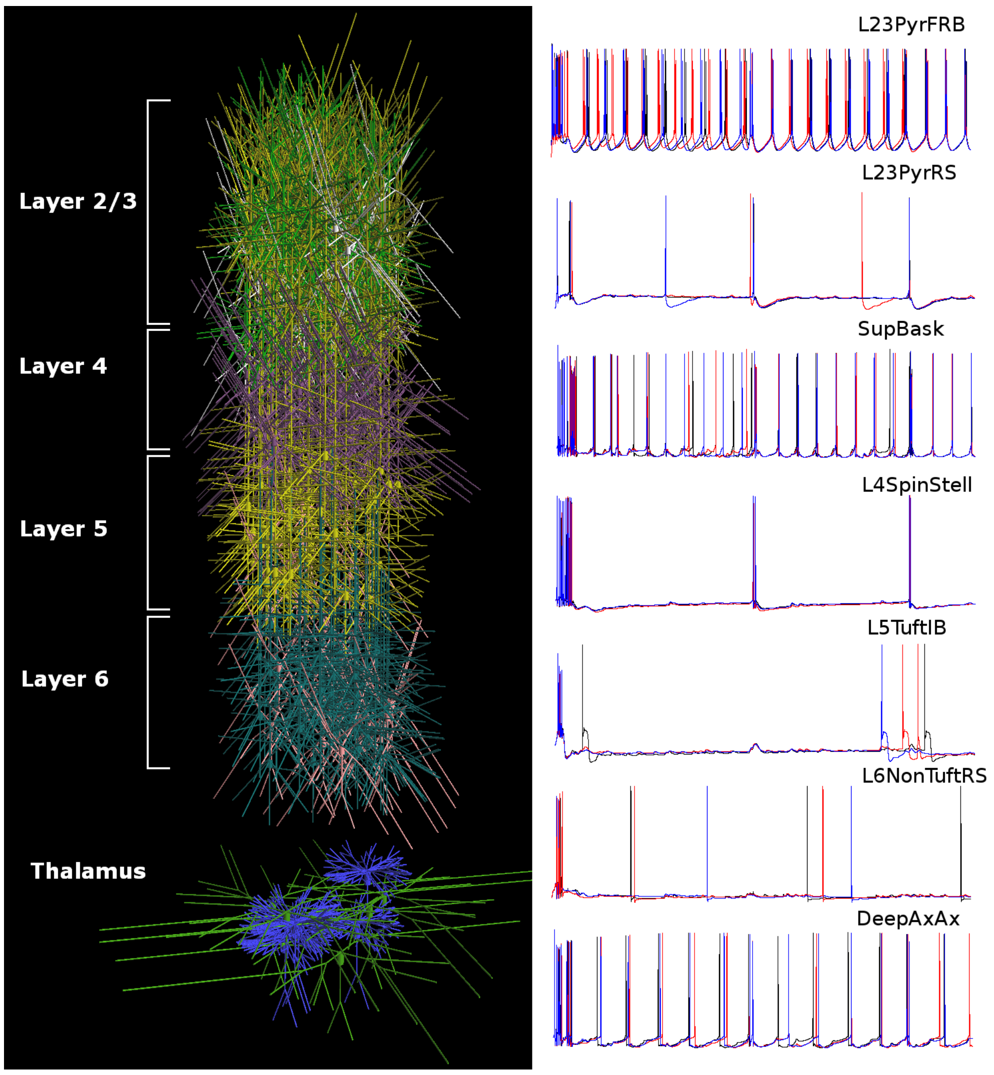


Figure 5.15: Traub et al (2005) thalamocortical network model extended to 3D. A) Visualisation of 3D network structure. B) Typical traces from a network simulation. Illustrated are membrane potential plots from a number of cell types (3 instances each) from a 1 second network simulation, using the network configuration from Figure 6 in Traub et al (2005).

efficiently optimise cell models against experimental data. The inclusion of gap junctions between almost all cell types (Figure 5.14) plays a significant part in the synchronous activity of the network, and is controversial with some experimentalists.

This model is an ideal candidate for open, collaborative model development by multiple interested labs, who can contribute extra data from their area of expertise and use different network configurations to address specific research questions. It is one of the initial models being used in our open source model development repository, as discussed in chapter 7.

5.3 Models from other brain regions

In addition to neocortical and cerebellar cell and network models, a number of other published models have been converted for use in neuroConstruct. Biophysically detailed modelling of the hippocampal formation is an active field (Cutsuridis et al, 2010), and a detailed CA1 cell model and a network model of the dentate gyrus are discussed below.

5.3.1 CA1 pyramidal cell model

This cell model was developed to investigate the backward and forward propagation of action potentials in the oblique dendrites of CA1 pyramidal cells (Migliore et al, 2005). It proved useful as a test case for conversion to NeuroML as it featured $\mathrm{I_h}$ and $\mathrm{K_A}$ currents with conductances which increased with distance from the soma. Also, the channel models used were supported by a range of simulators which had different approaches to solving multicompartmental cell models.

The model was originally developed in NEURON and that application's ModelView tool was used to export the cell in NeuroML Level 2. The

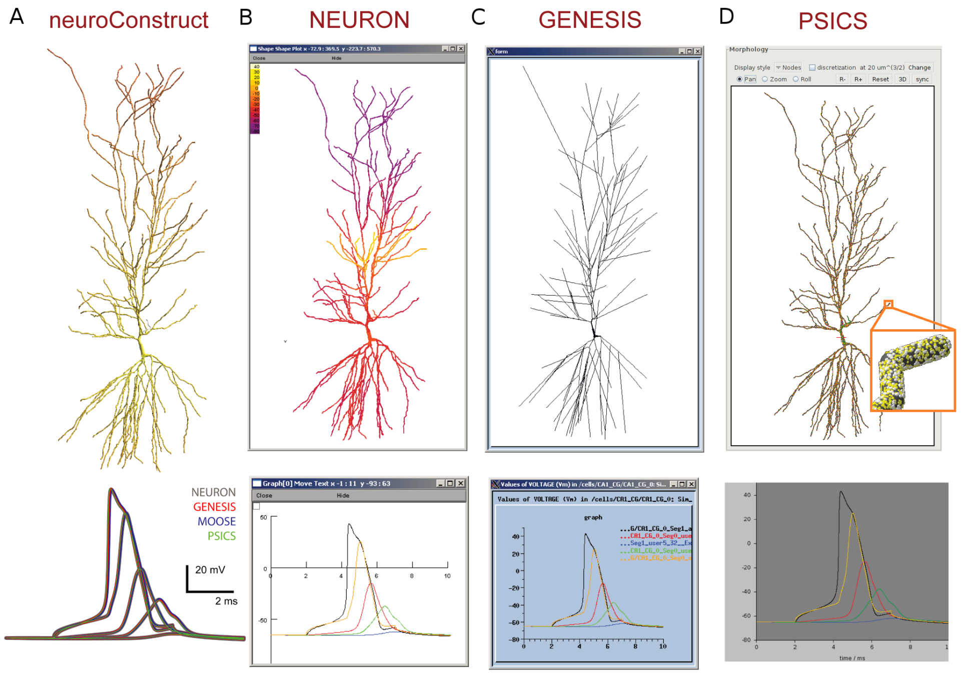


Figure 5.16: CA1 pyramidal cell model with non-uniform active conductances. A) Top: cell morphology visualized in neuroConstruct with color scale showing the density of h-type (HCN) channels (yellow lower, red higher). Bottom: voltage traces (in response to a current pulse input at the soma) at 5 different locations in the cell after execution on NEURON (gray), GENESIS (red), MOOSE (blue) and PSICS (green). B) Voltage map of same cell executed on the NEURON simulator (top) and membrane potential traces (bottom) for the axon (black), soma (yellow) and 3 locations (green, blue, red) at increasing distances along the dendritic tree. C) Re-compartmentalized morphology visualized and run in GENESIS (top) with membrane potential traces (bottom, colors as for panel B). D) Cell morphology visualized in PSICS using the ICING application (http://psics.org/icing, top). Inset shows a small section of dendrite and the locations of the individual ion channels. Membrane potential traces obtained with PSICS below, with colours as for panel B.

cell was imported into neuroConstruct and the distance dependence of the conductance densities for the nonlinear channels (which were converted to ChannelML) were set through the GUI. The detailed 3D cell and its response to a brief current injection in the soma are shown in Figure 5.16. The time courses of the membrane potential at various points along the cell was directly compared for the four simulators, NEURON, GENESIS, MOOSE and PSICS (Figure 5.16A, bottom). Despite important differences in the way each simulator handles the simulation of the cell anatomy and channels (e.g. the morphology was mapped to a reduced number of compartments on GENESIS/MOOSE, and the numbers of ion channels and their individual positions were explicitly calculated in PSICS), the physiologically measurable output of the cell was very similar across all simulators tested (Figure 5.16B-D) confirming the simulator independence of the NeuroML model description on short timescales and for a realistic neuronal morphology.

5.3.2 Dentate Gyrus network model

This is a key example of a model of a neural system transitioning from healthy to unhealthy activity, based on established physiological alterations. It was developed to investigate the contribution of mossy cell loss and mossy fibre sprouting to hyperexcitability in the dentate gyrus (Santhakumar et al, 2005). The model consisted of four cell types (granule, basket, mossy and hilar cells) with small numbers of compartments (9-17) and 10 types of active membrane conductance. Connectivity was based on experimentally measured values for synaptic properties and convergence and divergence of connections between cell types. Network activity was investigated for baseline connectivity, and with varying degrees of mossy fibre sprouting (recur-

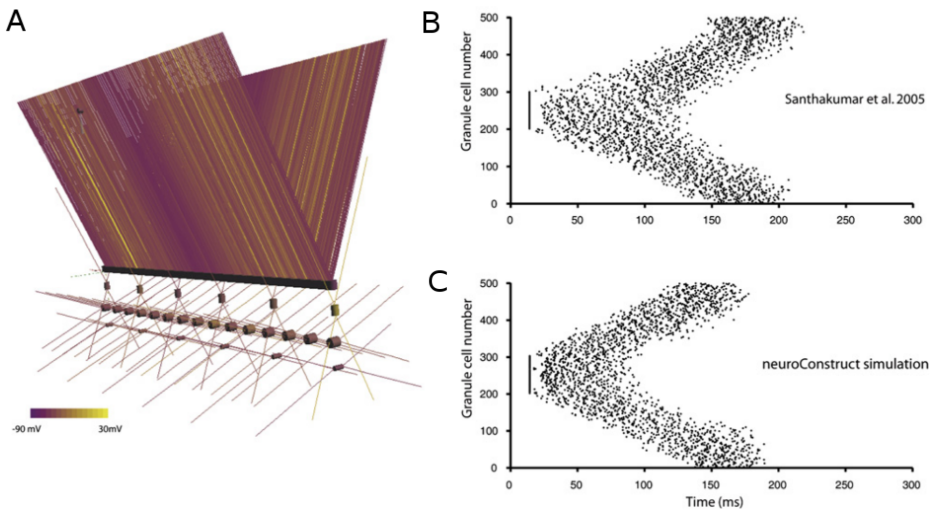


Figure 5.17: Dentate gyrus model of Santhakumar et al (2005). A) Replication of a network model in neuroConstruct. The model consists of (from the top down) 500 granule cells with two dendritic branches, 6 basket cells, 15 mossy cells and 6 hilar cells. The 10,000+ synaptic connections have been removed for clarity. The network receives a brief perforant path focal stimulation, mainly on the central 100 granule cells. Cell colouring reflects network activity 110 ms after stimulation. B) Raster plots of dentate gyrus granule cell activity in network with 10% mossy fibre sprouting in the original published model as obtained from ModelDB. C) Activity in the neuroConstruct implementation of the network.

rent excitatory connections via granule cell axons) or mossy cell loss. These changes have been associated with head trauma and hyperexcitability in the dentate gyrus. The network model also shows this increased excitability in both these cases.

The cells were manually reconstructed in neuroConstruct through the GUI and the original NEURON mod files were used for the channel mechanisms. This was one of the first network models to be converted to neuroConstruct to test its functionality and not all of the channel mechanisms have yet been converted to ChannelML. The network was recreated using the connectivity data from the original model (Figure 5.17A). The behaviour of the model in the 10% sprouted case after perforant path stimulation is similar for the neuroConstruct generated simulation in NEURON and the original model (Figure 5.17B, C).

5.4 Conclusions

The cell and network models presented here cover a wide range of brain regions and vary in detail from single compartment cells to multicellular large scale network models. In addition to use of some of these models in ongoing research (Rothman et al, 2009; Vervaeke et al, 2010) converting these models to neuroConstruct and NeuroML has served a number of important purposes. Firstly, they provide proof of principle that NeuroML is capable of representing the types of conductance based cell and network models being actively developed by researchers today. Manual conversion to NeuroML of the original NEURON and GENESIS scripts, developed by various modellers, each with their own programming styles, has helped me ensure that the language is flexible enough for a broad range of the channel, synapse and cell types and modelling formalisms.

Secondly, they have illustrated how neuroConstruct can extend existing network models to incorporate 3D features which would have been difficult to add in the original scripting languages. Anatomically realistic cell densities and layouts, cell region specific connectivity and accurate axonal innervation can be incorporated, and importantly, the networks can be visually inspected and analysed to ensure the correct structure and to compare the generated networks to their biological equivalents.

Finally, testing the models across multiple simulators has highlighted important issues about how precise model behaviour can be dependent on the numerical implementation algorithms used by individual simulators. An important part of sharing and reusing models (and so benefiting from the large amount of work that goes into making them) will be exchanging them a form that can produce stable results across many platforms without having to use excessive spatial discretisation or very small timesteps. Making well tested models available in NeuroML will be a crucial step towards this.

Chapter 6

Validation and Evaluation

Previous chapters have outlined work undertaken in developing new tools and standards for computational neuroscience, and have described a number of biophysically detailed cell and network models which can be used in conjunction with the new applications. In this chapter I will evaluate how well the solutions presented have met the original requirements for their development. The objectives of the work will be summarised and these will be used to evaluate how well the anticipated functionality has been achieved.

6.1 Validation criteria

To quantify how well the functionality of neuroConstruct and the opportunities afforded by NeuroML have met the original requirements, it is useful to summarise the objectives which have driven this work.

I) Simulator independent representation of complex neuronal structures: Simulators and other applications for handling detailed neuronal morphologies generally use their own internal structures for representing cells and this is has resulted in multiple incompatible file formats for morphologies. A better solution would be to create a superset of the target formats, which could be transformed from one specific representation into another.

II) Facilitating the generation of biologically realistic network connectivity in 3D: While the scripting interfaces of available simulators allow some 3D positioning and connectivity generation, more functionality is needed to recreate the complex layouts of cell populations and cell region specific connectivity found in real neuronal microcircuits such as those found in the cortex and cerebellum.

III) Cross simulator validation of cell spiking behaviour: Multi-compartmental, conductance based cell models are very complex and can be prone to errors in specification, incorrect or suboptimal implementation on a given simulator, or even bugs in the simulators themselves. A specification of the cell model in terms of its biophysical parameters independent of any given simulator and the ability to quantitatively compare model behaviour across implementations are crucial steps towards enabling greater sharing of these models between researchers. Better tested, purely physiological descriptions of model components will increase their transparency and ensure simulations are more faithful to the underlying biological mechanisms.

IV) Support to allow comparison of models at different levels of description: Models of spiking behaviour of a given cell type can be represented at many different levels, from point neuron models to highly complex representations with active dendrites. Different research questions will require different modelling approaches. The tools produced should make it easier to create models at any of these levels, compare them directly and identify the appropriate level of description to address a given hypothesis.

V) Comparison of existing models with those extended to 3D: Most network models published to date have been one or two dimensional representations of biological networks (or have ignored spatial aspects entirely). More realistic 3D representations of the networks which incorporate anatomically derived connectivity, should enable new scientific questions to be addressed.

VI) Enabling network simulations at scales closer to biological networks: Large scale network models using simplified point (usually leaky integrate and fire) neurons can approach anatomically realistic levels of cell numbers and connectivity. This is not normally the case with networks containing multicompartmental conductance based neurons, leading to simplifications in the numbers of synaptic inputs to each cell and reduced overall cell count. To investigate whether this will have an effect on network behaviour, much larger scale versions of such networks should be simulated, requiring new tool support.

VII) Ability to express a range of cells and microcircuits from different brain regions: The tools developed should have a broad applicability to the range of systems studied in the nervous system, and support the diverse models produced used by independently working researchers.

6.2 Evaluation

Comparison of the performance of the solutions presented in this thesis against the original objectives is presented below.

6.2.1 Fulfilment of objectives

I) Simulator independent representation of complex neuronal structures: The internal representation of neuronal morphologies as used in neuroConstruct, which is closely linked to MorphML, can be mapped to multiple formats (Crook et al (2007), section 2.2.1, Figure 2.3), both for import (e.g. SWC/Cvapp or Neurolucida format) and export (NEURON, GENESIS, PSICS, see Figure 5.16). The morphological descriptions can be accompanied by information on the non-uniform distribution of ion channel conductances across the cell membrane with Level 2 of NeuroML (Gleeson et al (2010a), section 3.3.2). Morphological representations of detailed cells in neuroConstruct can contain the full information on all 3D points as obtained in Neurolucida reconstructions. This is often more detail than required for compartmental models of the neuron. neuroConstruct has functionality for generating reduced representations of such cells which will lead to more efficient simulations (Gleeson et al (2007), section 2.2.1). The validity of the simulator independent NeuroML representations of cells and the various transformations produced by neuroConstruct was illustrated with the convergence of cell spiking behaviour when sufficiently large numbers of compartments were used (Figure 5.12, see also point III below).

A number of other applications for archiving, visualising or generating neuronal morphologies have added export features for NeuroML, including NeuroMorpho.org (section 3.5.9), the TREES Toolbox (section 3.5.10), CX3D (section 3.5.11) and NETMORPH (section 3.5.12). While I have been directly involved in adding NeuroML support to these applications, an increasing number of tool developers have independently added NeuroML support[1], illustrating the general applicability of the format and usefulness to the community.

Other formats for sharing neuronal morphologies such as SWC format of Cvapp and Neurolucida are still widely used, but the novel features of the

[1] http://www.neuroml.org/tool_support

NeuroML approach include the ability to closely link the cell structure to descriptions of biophysical properties, and the use of structured metadata related to the reconstructed cell.

II) Facilitating the generation of biologically realistic network connectivity in 3D: The fundamentally 3D nature of the cells in neuroConstruct has facilitated integration of anatomically inspired cell body placement and connectivity generation into the application's functionality. Cell body placement in defined 3D regions has allowed network models to be created with realistic densities of simplified cell models (e.g. of the cerebellar granule cell layer, Figure 4.3), and more detailed models that include full dendritic morphologies (Vervaeke et al (2010), Figure 5.7D).

The two main types of connectivity algorithms, Morphology Based and Volume Based (Figure 2.6) have been used to connect cerebellar (Gleeson et al (2007), Figure 5.5) and cortical (Figure 5.13, Figure 5.15) networks in 3D. Connection probabilities between cells in a network can be complex functions of inter cell distance. Experimental measurements of these properties have been included in a detailed 3D network model of Golgi cell electrical coupling (Vervaeke et al (2010), Figure 5.7A, B).

Network generation in neuroConstruct is based on repeated instances of morphologically identical 3D cells within each population. There are a number of options to allow heterogeneity in spiking behaviour within populations (e.g. Vervaeke et al (2010) used variable input resistances to modify the intrinsic firing frequencies of each Golgi cell). An alternative approach would be to generate unique morphologies for each cell in the populations according to a defined set of rules. This has been the approach taken with a number of other tools, though creating a software package to simulate neuronal elongation/branching (as used in NETMORPH), growth towards chemoattractors (e.g. CX3D) or generating new neurons by sampling branch points in reconstructed cells (e.g. TREES Toolbox) are complex research tasks in themselves. My approach has been to push for interoperability with these tools through NeuroML to integrate such networks into the neuroConstruct toolchain.

III) Cross simulator validation of cell spiking behaviour: The development of a simulator independent format for expressing voltage and ligand dependent conductance models and their distribution across cell membranes has enabled quantitative comparison of the behaviour of complex spiking cell models across simulators for the first time. However, simulators take different approaches to the numerical integration of the equations describing the models and have different methods of compartmentalising complex cells. An important outcome of this work has been to show that convergence in spiking behaviour can be achieved with 1) a sufficiently small integration timestep and 2) a fine spatial discretisation. Figure 5.11A shows a single compartment cell model with channels from Traub et al (2005) with a commonly used timestep of 0.025 ms on the same three simulators for a 100 ms simulation. Spike times match to within 0.1 ms. It also illustrates how not just the membrane potential, but also the internal channel variables are matched between simulators. Figure 5.2D shows the convergence of a single compartment granule cell model on NEURON, MOOSE and GENESIS as the timestep for numerical integration becomes smaller. A time step of 0.001 ms is required to ensure spike times on all simulators match to within 0.1 ms over a simulation run of 600 ms, showing how subtle differences in spike times between simulators can occur for longer simulations, unless a sufficiently small integration time step is used.

For multicompartmental cell models, convergence in spiking behaviour across simulator implementations can also be achieved, Figure 5.16 shows a detailed CA1 model run across four simulators, while Figure 5.9 and Figure 5.4 show a layer 5 pyramidal cell and a Purkinje cell respectively, executed on both NEURON and GENESIS. The convergence of spiking behaviour when spatial discretisation is increased is demonstrated in Figure 5.12. This figure highlights how different complements and distributions of active conductances can alter the level of spatial discretisation needed in a cell model before its behaviour can be said to be independent of the number of spatial integration points used.

Quantitative comparison of the differences in behaviour of the cell models between simulators is important. This was an integral part of the automated tests carried out on neuroConstruct projects (section 4.5). Simulations were run on all supported simulators for a given model and checks were made that all spike timings were within a given margin (normally less than 0.1 ms for simulations of hundreds of milliseconds). This automated testing ensures consistent model behaviour across versions of neuroConstruct and simulators.

The process of conversion of models to neuroConstruct and NeuroML and testing across simulators has helped identify errors in model implementations and even in simulators. An issue with the original GENESIS implementation of the Purkinje cell model (section 5.1.3) was highlighted (in collaboration with Arnd Roth) when it was tested on NEURON[2]. Incorrect

[2]This involved the use of tables of precomputed values for the voltage and [Ca^{2+}] dependence of the BK conductance. The [Ca^{2+}] dependence used too coarse a discretisation of [Ca^{2+}] values, leading to incorrect rate values for that conductance, subtly altering the firing behaviour. This error was reproduced with a customised NEURON NMODL file, a correct implementation in NMODL was produced, and the equivalent channel in Chan-

initialisation of the rate variables in the initial NEURON implementation of the channel models from Traub et al (2005) could not be implemented in ChannelML, and so the correct form had to be used[3]. Tests on the discretisation of the cell models from that same publication (Figure 5.12) helped highlight a bug in GENESIS which required a finer spatial discretisation to be used for that simulator[4].

IV) Support to allow comparison of models at different levels of description: Cell models in neuroConstruct and NeuroML can be created as point neuron models, abstract representations of cell structure with a handful of compartments or morphologically detailed models. An example of this can be seen in Figure 5.3 where Golgi cell models at three levels of abstraction are shown. Each of these has a set of active conductances on the soma and varying detail for the (passive) dendritic and axonal trees. Cell anatomical and biophysical properties can be compared easily, as can spiking behaviour (similar cell activity in response to hyperpolarising and depolarising current injections are shown). Koen Vervaeke developed the

nelML created (all versions of this channel are present in the neuroConstruct project for this cell). The ChannelML implementation was used to produce the matching traces in Figure 5.4. This bug would have been very difficult to isolate in a single simulator implementation.

[3]The values for m, h, etc. were initialised to zero as opposed to the correct value at resting membrane potential, to reproduce the original FORTRAN implementation.

[4]In GENESIS, there are two options for simulating compartments: symmetric compartments where the axial resistance is divided in two at each end of the compartment and the voltage is effectively calculated at the centre of the compartment, and asymmetrical compartments, where all of the axial resistance is at one side and which are slightly more efficient to use in simulations. Ideally for matching the behaviour of NEURON and GENESIS, symmetrical compartments should be used, as NEURON also calculates the voltage at the centres of *nseg* regions of the section. However, in GENESIS v2.3 there is a known bug (personal communication with Hugo Cornelis and David Beeman) which doesn't allow use of the hsolve numerical integration method with symmetrical compartments when there are compartments with more than 2 child compartments (e.g. soma of cells in Figure 5.11B-E with multiple dendrites). Use of hsolve is required for simulations as this is much faster than the basic Exponential Euler method. That method would require a much smaller dt ($\sim$0.00001 ms) for convergence of the simulation. Therefore, in the simulations discussed here asymmetric compartments, together with hsolve are used. This necessitated a much finer spatial discretisation of the cells (Figure 5.12).

model illustrated in Figure 5.3C from that shown in B using the latter's implementation in neuroConstruct and an imported Neurolucida reconstruction. With models at multiple scales available, researchers can choose the appropriate detail of model to use in their work, e.g. single compartment representation in a large scale network (Figure 4.3), detailed model when signal propagation in dendritic trees is relevant (Figure 5.7D).

NeuroML v2.0 affords even greater possibilities for creating both reduced representations of cell models, e.g. one or two variable abstract neuron models (section 3.6), and more complex cells, which include subcellular signalling pathways (section 4.6). Researchers developing at either of these levels can compare their cells' behaviour to examples of primarily conductance based cell models already available, which will help improve overall electrophysiological realism of the models at each scale.

V) Comparison of existing models with those extended to 3D: Creating 3D network models in neuroConstruct allows more direct comparison of models that omit and include anatomically realistic spatial representations. The extension to 3D of the originally 1D model of Maex and Schutter (1998) (Gleeson et al (2007), section 5.1.4) allowed comparison of Golgi cell synchronisation along and across parallel fibre tracts in the model with recordings made in the cerebellum of anaesthetized rats (Vos et al, 1999). Use of 3D cell placement and experimental data on connection probabilities and coupling coefficients in the electrically coupled Golgi cell model (section 5.1.4) has allowed investigation of the spatial spread of network desynchronisation due to sparse synaptic input (Vervaeke et al, 2010), which would have been difficult with a one or two dimensional model.

The distribution of random synaptic inputs on a detailed layer 5 pyramidal cell (section 5.2.3) allowed demonstration of the principle of multiplicative gain modulation in a morphologically complex neuronal model, extending the findings in simple neuron models and experimentally in cerebellar granule cells in Rothman et al (2009).

While the current 3D version of Traub et al (2005) does not include distance dependent connection probabilities, it has been used for making more anatomically realistic models of cells placed in a cortical column (Figure 5.15), by matching known cell body placement, layer depths and dendritic lengths of the cells, something that would have been very difficult with the original 1D FORTRAN model.

VI) Enabling network simulations at scales closer to biological networks: The ability of neuroConstruct to automatically generate scripts for the parallel version of NEURON has allowed large scale network models to be executed, which feature multicompartmental conductance based models and the complex 3D connectivity as used in smaller, serial simulations. A cerebellar granule cell layer network containing one million single compartment cells has been tested (Figure 4.3), with performance of the generated code scaling approximately linearly up to 200 processors. A cortical column network with multiple populations of detailed cells has also been tested, with simulations run containing ten thousand cells, roughly on the order of a single anatomical cortical column (Figure 4.4). To make sure that such large scale simulations are physiologically plausible and well constrained, it is essential that the components are expressed in an accessible and cross simulator format and that connectivity properties can be readily checked (Figure 5.14).

Another advantage of neuroConstruct's automatic generation of (large scale) network models and interaction with remote computing resources (section 4.2.4) is that it facilitates generation of families of models, e.g. to test the robustness of models against key parameters, or to create multiple instances of stochastic models, to allow error bars on graphs depicting network behaviour (see Figure 8 of Vervaeke et al (2010)).

More work can be done to improve the parallel code generated by neuroConstruct to take advantage of recent features added to Parallel NEURON, including splitting of large cells across multiple processors, which will help balance networks with a mixture of complex and simple cells, e.g. Figure 5.8.

VII) Ability to express a range of cells and microcircuits from different brain regions: The range of cell and network models which have been converted for use with neuroConstruct and NeuroML include examples from the cortex, cerebellum and hippocampus (chapter 5), which were originally developed by a number of different labs in a range of languages. The original target network of neuroConstruct was the cerebellum, and initial connection generation functionality was heavily influenced by the stereotypical arrangement of synaptic connectivity in that region, including between parallel fibres and Purkinje cell dendritic trees. However, these connection mechanisms (Figure 2.6) have proven versatile enough for other brain regions.

Initial work has taken place converting models from other brain regions into NeuroML, including the granule and mitral cells of the olfactory cortex. There is also a project under way to convert the entire connectome of the nervous system of Caenorhabditis elegans to pure NeuroML[5]. NeuroML version 2.0 opens up new possibilities for interactions between neuronal net-

[5]http://code.google.com/p/openworm/

works and non neuronal model elements, e.g. astrocyte modelling Figure 4.9.

6.2.2 Conclusions

As outlined, the majority of the original objectives of the project have been fulfilled. Models expressed in NeuroML can be loaded into neuroConstruct and run across multiple simulators, with quantifiable comparison of behaviour. More complex network models can now be created in 3D of multiple neuronal microcircuits, and a number of publications based on these models have been produced.

Nevertheless, the cell and network models presented here represent just a small fraction of the published models available to researchers through resources such as ModelDB. Models of channels, synapses and neuronal morphologies are used in many different scenarios in neuroscience research, not merely for detailed compartmental modelling. It is hoped that the tools and languages developed will have wider applicability beyond the range of uses we have put them to. Initial indications for this are good, with a number of publications coming out recently which use neuroConstruct for original research (section 7.1.5), and a number of independently developed tools supporting NeuroML[6].

Specific shortcomings of the solutions presented here include the relative lack of export functionality for NeuroML from supporting applications. It would be good to be able to export a fully compliant model with cell and channel descriptions from NEURON for example, and import it natively into MOOSE, without requiring an intermediate step like neuroConstruct. This functionality is part of the long term plans of many of the developers

[6]http://www.neuroml.org/tool_support

involved. Another issue with the current implementation of neuroConstruct is that while randomly connected networks are quick to generate, every extra condition on a network connection (distance dependent connection probability, limiting numbers of presynaptic connections) will increase generation time. This is not a problem for most networks but for larger scale networks it will be a trade off between incorporation of full details and time spent generating the networks. Work is continuing in close cooperation with other developers to improve the overall performance of neuroConstruct and to optimise generation and simulation of the core set of example models. More details on the future plans related to neuroConstruct and NeuroML are discussed in chapter 7.

Chapter 7

Discussion

In this thesis, I have outlined the development of a number of tools and interoperability initiatives which facilitate the creation, simulation, analysis and exchange of detailed cell and network models. This has been motivated by the difficulty in creating neuronal network models with anatomically realistic 3D structure and connectivity. Also, the disparate set of software tools available for creating such models which have multiple incompatible formats, have made them difficult for other researchers to use, critique and build on. Key design and implementation considerations of this work have included making the tools and standards as accessible and as open as possible, towards creating a solid foundation for a software infrastructure for biologically detailed models of brain function.

7.1 Facilitating creation and analysis of 3D network models

neuroConstruct has been designed from the start to facilitate creation of models of neurons and networks which incorporate a high degree of anatomical and biophysical detail (Gleeson et al (2007, 2008), chapter 2). The application is inherently three dimensional, with fully reconstructed neuronal morphologies and explicit 3D cell placement and connectivity at its core. This has enabled creation of models of neuronal systems incorporating a level of anatomical and physiological realism which has not been possible before.

7.1.1 Enabling more realistic network models in 3D

The spiking activity of many cell types depends crucially on their morphologies and compliment of active membrane conductances (Johnston and Narayanan, 2008; Migliore and Shepherd, 2002). Creating multicompartmental, conductance based cell models incorporating realistic neuronal morphologies and nonuniform active conductances is a key feature of neuroConstruct. These can be visualised, validated and the cell structures and biophysical properties manually adjusted through the graphical interface in a much more user friendly manner than possible with previous solutions. Support for NeuroML allows the cell models to incorporate a wide array of voltage and ligand gated conductances, both Hodgkin Huxley like and kinetic scheme based, along with complex subcellular Ca^{2+} dynamics.

Quantitative anatomical measurements of network properties including cell densities, numbers of synaptic connections between cell groups and dimensions of axonal and dendritic fields are available for several brain regions including the cortex (Douglas and Martin, 2004; Somogyi et al, 1998), cerebellum (Eccles et al, 1967; Harvey and Napper, 1991) and hippocampus (Cutsuridis et al, 2010; Klausberger and Somogyi, 2008). However, generating biologically realistic 3D neuronal network models from such data has proved difficult using the direct scripting approach traditionally used in neuronal simulators. This is because, unlike many random artificial networks, networks of neurons in the brain exhibit inhomogeneous connectivity probabilities (Lübke et al, 2003), spatial clustering and an enhanced probability of certain multi-cell motifs (Song et al, 2005; Sporns and Kotter, 2004).

Several of the core functions I have developed for neuroConstruct facilitate the generation of 3D network models with increased biologically realism. These include the ability to import neuronal reconstructions in multiple file formats and the automated placement of cells in defined 3D patterns. Two algorithms enable synaptic connectivity to be generated in 3D space with subcellular specificity (Figure 2.6). The first was designed for cell models with fully reconstructed axons, axons that are rather invariant (e.g. parallel fiber-Purkinje cell and Schaffer collateral-CA1 synaptic connections (Shepherd, 2004)) and large terminals that innervate many postsynaptic cells (e.g. cerebellar mossy fibres). The second is designed for cells with dense axonal arborizations that project over a particular region of 3D space (e.g. spiny stellate cells in cortex (Lübke et al, 2003) and various interneurons in cortex, hippocampus and cerebellum (Shepherd, 2004)).

While many network simulations developed today use simplified representations of excitatory and inhibitory synapses (e.g. alpha function synapse), neuroConstruct allows a range of more physiological synapse models to be used in generated network connections. These include linear synapses with multiple decay time courses (AMPA, $\mathrm{GABA_A}$), voltage dependent synapses (NMDA), plastic synapses (exhibiting STP or STDP) and electrical synapses. Greater flexibility is afforded with support for synapses in NeuroML v2.0, allowing users to create customised synaptic plasticity models, for example.

Non-uniform network connectivity can be implemented in neuroConstruct by defining multiple groups of cells and/or connections and by functions that define the mean spatial dependence of connection probability or strength of coupling. This allows local circuits with spatially correlated synaptic connectivity, feed-forward inhibitory networks (Yoshimura and Callaway, 2005) and networks with "small world" properties (Watts and Strogatz, 1998) to be created. Also, highly skewed distributions of synaptic weights (Song et al, 2005) can be implemented.

In addition to more realistic connectivity patterns, the ability to visualise and execute simulations of large scale networks allows networks with anatomically realistic cell densities to be created. This feature, together with the complex connectivity patterns outlined above has been used in the electrically connected Golgi cell network model (Vervaeke et al, 2010) discussed in section 5.1.3. This model incorporated morphologically detailed, conductance based Golgi cell models based on single cell recordings, data on the cellular densities of Golgi cells in the cerebellar granule cell layer, and experimentally measured properties of electrical connectivity, including how probability of such connections and coupling coefficients vary with intersoma distance. In addition to the random cell placement and network connectivity properties, nonuniform cell properties were applied and cells received stochastic synaptic inputs. To investigate the behaviour of the model, families of model instances were generated through the GUI, which were run on remote high performance computing hardware. This model provided valuable insight into how gap junction coupling can be inhibitory, with the Golgi cell's large afterhyperpolarisation being preferentially transmitted. The spatial spread of the network desynchronisation caused by localised synaptic input could also be studied.

Creation and management of such a detailed network with existing script based simulators would have been very difficult, but projects developed within neuroConstruct can incorporate and easily manage this complexity. Creating and distributing the model in this way also makes it available for other researchers to reproduce the results, investigate all of the physiological parameters used and extend with their own new data, all through the graphical interface.

7.1.2 Accessibility of models

The graphical aspect of neuroConstruct is a key way to make the models more accessible to non computational researchers. Two dimensional neuronal microcircuit "wiring diagrams" (Grillner et al, 2005; Klausberger and Somogyi, 2008; Somogyi et al, 1998; Zeeuw et al, 2008) are commonly used by anatomists to communicate complex synaptic connectivity patterns. 3D graphical representations of such connectivity as afforded by neuroConstruct allows communication of a wider range of network properties between experimentalists, modellers and theoreticians. Network visualisation, together with other automatically generated views of structure (e.g. network connectivity matrix in Figure 5.14) are useful, but if these are closely linked to executable models, it makes a powerful framework for consolidating knowledge and testing theories of microcircuit function.

Another benefit of the application is the ability to generate simulations for complex neuronal simulation platforms without the need to learn their scripting languages. While there are many possibilities afforded to advanced users with the option to add custom native code to the simulations or to interact with neuroConstruct through the Python interface, all neuroConstruct projects can illustrate interesting model behaviour accessible through the graphical interface alone. This can benefit not just experimentalists interested in modelling, but also researchers and software developers outside the field who want to get a feeling for realistic neuronal modelling, without the need to learn a simulator specific programming language.

neuroConstruct also facilitates comparison of experimental and simulation data. Both loading in of experimental data through the GUI, and reading larger datasets using the Python interface make it easier to quantitatively compare experimental traces with simulations. Analysis techniques for experimental data can be reused on simulation results (section 2.3.2) further lowering the barrier for comparing real data and simulations (compare Figures 1C/D and 7E in (Vervaeke et al, 2010) and Figures 1d and 5b/d in (Rothman et al, 2009)).

7.1.3 Sharing model components

Biophysically detailed models are difficult to build and it can take 1-2 years to go from initial experiment to published model. It is therefore important that researchers do not have to repeat the same procedures due to the lack of accessibility of existing models. This can be avoided by reusing components from existing models, and is currently enabled with a simulator specific files from ModelDB. It will be increasingly important in the future that these components are stable, documented and well tested. neuroConstruct has a number of features for sharing of model components, both between projects and for exporting to standardised formats, and for checking their validity. Increasing automation of this will make it quickly clear when a component imported into a model may give spurious results (e.g. a channel model designed for use at one temperature being used at a different one).

Sharing of model components and reusing them across simulators will also increase the overall quality of the components. Detailed neuronal models are complex software entities prone to bugs and simulator dependent behaviour (see point III in section 6.2.1 for examples). Making it easy for modellers to test their models across many simulators will help to eliminate these.

7.1.4 Technology integration

neuroConstruct can also serve as a technology integration platform. While the application can be used as a graphical user application on a desktop creating simulations for running locally or remotely, it can also be used as a library in a larger toolchain. The source code or compiled libraries can be integrated into other Java applications since the code is open source and well documented. Python too by its nature encourages gluing together different specialist modules and libraries. An application developer could for example include neuroConstruct as a module (e.g. to convert SWC files into NEURON format) and use the range of other modules in the standard Python distribution to generate web pages, email PDF reports, interact with databases to create a new neuroinformatics resource.

One example of such a scenario in use is the Whole Brain Catalog[1] project, which is using neuroConstruct on its backend systems to generate simulation data for neuronal networks constructed and visualised through its multi-scale graphical interface (see section 7.3.1).

Figure 7.1 shows the range of simulators and other applications which can currently interact with neuroConstruct, or which can import and/or export NeuroML. Many of these applications also have Python interfaces, and so a huge number of interesting combinations of these tools are possible in a data generation and analysis workflow.

[1]http://www.wholebraincatalog.org

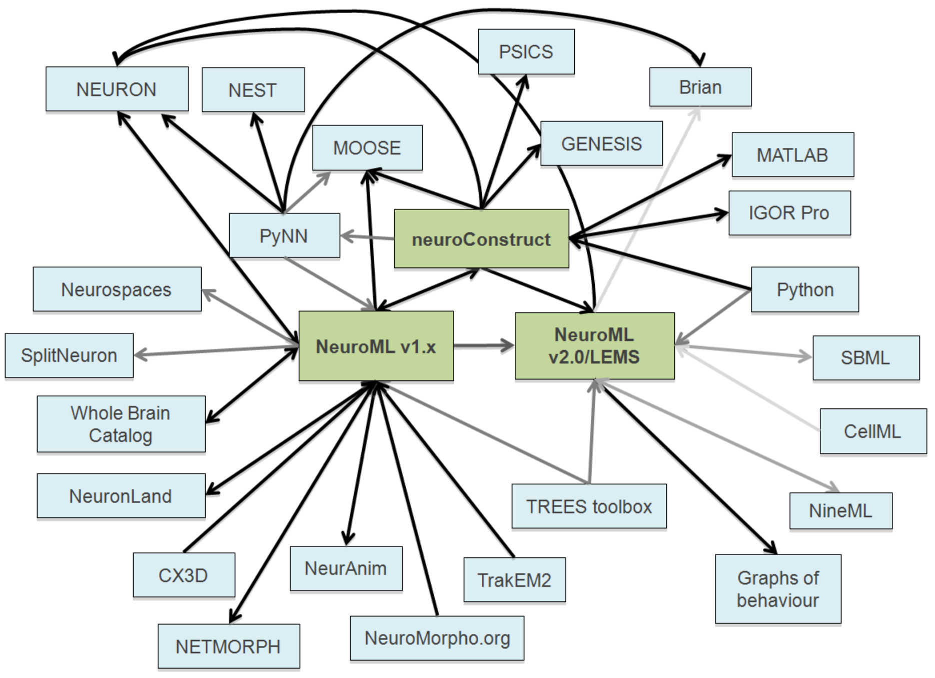
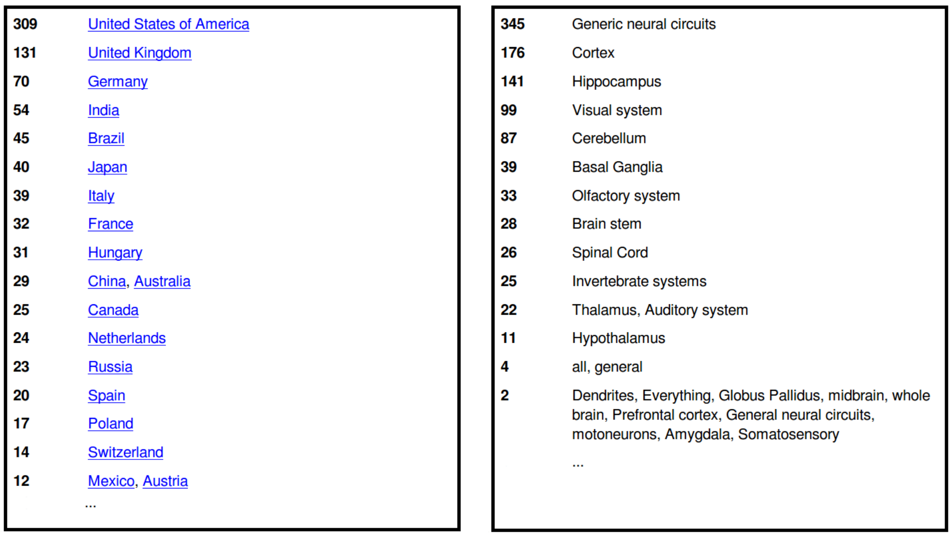


Figure 7.1: Software applications which can interact with neuroConstruct and NeuroML. The arrows indicate the direction of interaction (e.g. exporting to NeuroML is shown by an arrow to that format). Stable functionality is shown with black arrows, lighter colours indicate functionality in development. More details on all NeuroML compliant applications can be found at http://www.neuroml.org/tool_support.



309	United States of America
131	United Kingdom
70	Germany
54	India
45	Brazil
40	Japan
39	Italy
32	France
31	Hungary
29	China, Australia
25	Canada
24	Netherlands
23	Russia
20	Spain
17	Poland
14	Switzerland
12	Mexico, Austria
	...

345	Generic neural circuits
176	Cortex
141	Hippocampus
99	Visual system
87	Cerebellum
39	Basal Ganglia
33	Olfactory system
28	Brain stem
26	Spinal Cord
25	Invertebrate systems
22	Thalamus, Auditory system
11	Hypothalamus
4	all, general
2	Dendrites, Everything, Globus Pallidus, midbrain, whole brain, Prefrontal cortex, General neural circuits, motoneurons, Amygdala, Somatosensory
	...

Figure 7.2: Usage statistics for neuroConstruct, based on registration data for users downloading the application. Details on the country of origin (left) and the brain region of interest (right) of users. Lists have been truncated to show top responses.

7.1.5 Usage of neuroConstruct

neuroConstruct has been available for download (section 2.4.1) since 2007. As of September 2011 the application has been downloaded by over 1,100 registered users from over 40 countries (Figure 7.2).

In addition to the original scientific research which has used neuroConstruct within the Silver Lab (Rothman et al (2009); Vervaeke et al (2010); investigation of NMDA spike generation in layer 5 pyramidal cells (Farinella et al, 2011)), published research which has been carried out using neuroConstruct includes: investigation of the electrical properties of Drosophila neurons which utilises realistic morphological reconstructions and electrophysiological recordings (Gouwens and Wilson, 2009); investigation of the effects of stochastic L-Type Ca^{2+} channels on the response of SK channels in Ca^{2+} microdomains (Stanley et al, 2011); looking at the emergence of small world network properties of networks containing synapses with spike timing dependent plasticity (STDP, Basalyga et al (2011)); creating a biophysically detailed cortical column model to study the origin of the voltage sensitive dye imaging signal (Chemla and Chavane, 2010); investigation how synaptic pathologies (alteration in long term depression (LTD)/potentiation (LTP), inhibition or connectivity) can underlie specific cognitive impairments (Hanson and Madison, 2010). This list suggests that neuroConstruct has already been adopted by many groups around the world for a wide range of neurophysiological investigations.

7.2 Current state of standardisation and interoperability initiatives

Standardisation efforts towards greater data sharing and software interoperability in many scientific fields are well advanced. In biology, the bioinformatics and systems biology fields are leading the way with publicly accessible databases of well structured and annotated data and models (Le Novère et al, 2006; Lloyd et al, 2008). Standards for model description and exchange in neuroscience are maturing, and an increasing number of tool developers are making their software compliant to these to gain the full benefits interoperability affords.

A positive outcome of this is that researchers can focus more on their core strengths, making specialist tools for their area of interest. For example, developmental neuroscientists who create an algorithms for neuronal outgrowth (as in the cases of CX3D and NETMORPH, section 3.5) can concentrate on implementing their core functionality, and rely on other applications which support the same standards it does for visualisation, morphometric analysis and conversion to simulation scripts.

7.2.1 Evolution of the NeuroML initiative

Version 1.x

I have made a substantial technical contribution to the NeuroML initiative since becoming involved in it in 2005. Its scope covers the majority of conductance based neuronal models in use today (Crook et al, 2007; Gleeson et al, 2010a). As opposed to specifying the language in advance and waiting for tool developers and modellers to adopt it, it has been developed in parallel with the conversion of published cell and network models to the format, and in close collaboration with simulator developers.

This XML based language has a modular structure and v1.x is sufficiently advanced to allow the description of the complex branching structures of dendritic trees and axonal projections, their biophysical properties, voltage- and calcium-gated ion channels, chemical synapses with short-term synaptic plasticity, electrical synapses, and both large and small scale network structure. The implementation and interoperability of models expressed in NeuroML have been well tested and the functionality validated by expressing existing single neuron and network models of different brain regions in this format and by demonstrating equivalent model behaviour on different simulators (chapter 5).

Providing a structured, declarative framework for describing detailed neuronal models that is independent of any particular simulator implementation has a number of important benefits. Firstly, the behavioural properties of a model specified in NeuroML can be compared across simulators. This is important for testing the validity of results from a model, since all conclusions should be simulator-independent. Model comparison also aids bug identification, tests the robustness of a particular model implementation, highlights performance bottlenecks and promotes collaboration between different simulator communities. Secondly, describing model components with structured schemas written in XML facilitates machine automated validation of particular components (e.g. the integrity of a complex neuronal morphology defined in MorphML). Thirdly, the modular structure of NeuroML, and the standalone nature of many of the mechanisms, facilitates reuse of model components. This speeds up the construction of models and allows models with increasing biological detail to be built from previously developed components. Enabling interoperability will accelerate the rate of progress by allowing investigators to use and extend previous work, rather than "reinventing the wheel" each time they want to build a new model. Such models will also provide a ready-made resource for developing and testing new software tools in this area.

The set of models I have converted to NeuroML and the simulator mappings I have developed have helped highlight a number of technical issues which will be important for true cross simulator model development. My results show that a key reason why the spike times of some multicompartmental cell models can diverge between simulators is the different way they treat neuronal morphology and the different locations at which the voltage is computed within each compartment. While increasing spatial discretisation and decreasing time step lead to model convergence (Figure 5.2; Figure 5.12), for some cells this only occurred in computationally inefficient regimes. Such direct comparison of the performance of different simulators will allow the most efficient solution to be identified, potentially improving overall simulator implementations.

While I have developed a number of the current NeuroML compliant applications (section 3.5), independent development of NeuroML v1.x support is increasing (Figure 7.1), and now 22 software applications support some part of the language, and more have NeuroML support in their development roadmaps[2].

Version 2.0

The focus on multicompartmental, conductance based models in v1.x resulted in good cross simulator support for NeuroML by simulators which specialised in this type of modelling. However, one of the drawbacks of this version was that while the definitions of the behaviour of an ion channel or plastic synapse were present in the text of the specifications, a simulator had to natively support the concept of an active conductance, or the specific STP model to use that part of the language. It also made the language difficult to extend: NeuroML had to be updated at its core to add new model types.

Incorporating new model types is enabled by a mechanism in NeuroML v2.0 for providing explicit, machine readable definitions of model components. The full definitions of all channels, synapse models and abstract cells are specified in terms of their parameters and state variables and how these change with time, or on receiving a particular event (using LEMS, section 3.6.3). The standardised programming interface offered by libNeuroML (Figure 3.17) to access model component definitions will give simulators new opportunities for low level interaction with model definitions. Together with the hierarchical framework for neuroscience specific core model types in NeuroML v2.0 (Figure 3.15), the simulators will be provided with the information as to whether a component is a conductance based synapse or

[2]http://www.neuroml.org/tool_support

a kinetic scheme based ion channel and can optimise execution accordingly.

Many different levels of abstraction are used in neuronal modelling (Figure 1.1) and simplified one or two variable point neuron models are widely used for investigating properties of generic neuronal networks (Brette and Gerstner, 2005; Izhikevich, 2003) and are favoured by applications aimed at large scale network models (Diesmann and Gewaltig, 2002; Izhikevich and Edelman, 2008). Support for these types of abstract cell models will be enabled by this new model dynamics specification scheme and represents a key advance in the scope of NeuroML.

This new extensible system can be used to define not only electrical and chemical properties of cellular components. Physical systems can also be described (e.g. using x, y, z as exposed variables, or theta for joint angle or eye direction) which can interact with the neuronal components according to specified rules. This leads to intriguing possibilities for simulations incorporating interactions with the physical environment. These possibilities have already been discussed with groups interested in developing an *in silico* model of C. elegans[3], and researchers investigating the Xenopus tadpole nervous system through experiments and modelling[4].

An interface for parsing model structure over multiple scales will also help with automatic assignment of different time steps to different model components, facilitating multiscale modelling. Support in libNeuroML for mapping the structures of systems biology languages such as SBML and CellML to and from ComponentClasses will allow greater subcellular details to be incorporated into neuronal models, e.g. signalling pathways and gene regulatory networks.

Much of the framework of NeuroML v2.0 is already in place and it will

[3]http://code.google.com/p/openworm

[4]http://www.bristol.ac.uk/biology/research/behaviour/xenopus/

enable unprecedented sharing of model behaviour details and low level interoperability between computational neuroscience applications.

7.2.2 Improving model quality through cross simulator validation

Being able to run a model across multiple simulators also provides a crucial quality assurance mechanism for complex neuronal models. Even the most experienced modeller can introduce behaviour into their models which is dependent on the specific numerical integration methods of the simulator being used. The process of converting the model to NeuroML and trying to get identical behaviour on another simulator has uncovered inconsistencies with published model descriptions, bugs and bad practices in almost all of the models I have converted (section 6.2.1). Most are minor, but the key point is that a model should stand alone, independent of any particular instantiation in a simulator.

In addition to error checking, cross simulator testing of models can improve overall simulator quality by comparing their performance in simulating the same model. Friendly rivalry between the NEURON and GENESIS/MOOSE communities has lead to various optimisations of behaviour in each of the platforms, which was difficult when manual conversion of models between the native formats was required.

7.2.3 Community engagement

Discussions between modellers and simulator developers have helped highlight key concepts and bottlenecks and have greatly helped the NeuroML development process in recent years. Three NeuroML Development Workshops have been held to date, two of which I have been the main organiser.

These have been instrumental in encouraging uptake of v1.x and planning for NeuroML v2.0.

In addition, there have been active contacts between members of the NeuroML community and the INCF Program on Multiscale Modelling[5]. This Program set up a Task Force, of which I was part, to look at developing standards for describing large scale networks of spiking neurons. The specification which has emerged, NineML[6], broadly overlaps with NeuroML v2.0 and LEMS. It too has machine readable definitions of model components at its core. Efforts have already been made to use the same XML serialisation and object model for this common part (initially covering descriptions of abstract cells and synapses), and this drive towards low level compatibility will continue as both projects develop.

NineML has mainly been developed by the PyNN developers and focusses on large scale networks of simple neurons running on simulators like NEST, and NeuroML v2.0 has, from the start, required support for multicompartmental conductance based models and networks which were supported in v1.x. These different priorities will be beneficial in future, allowing NeuroML compliant tools like neuroConstruct to run more models on simulators like NEST through the Python based libNineML, and increase the biophysical detail of cell models which will be available for simulators like NEST and Brian.

The COMBINE (COmputational Modeling in BIology NEtwork) initiative[7] will be a forum for greater dialogue between groups interested in modelling in any area of biology. Formed initially by the SBML and BioPax[8] communities, it seeks to encourage open, community driven development of

[5]http://www.incf.org/programs/modeling
[6]http://software.incf.org/software/nineml
[7]http://www.co.mbine.org
[8]http://www.biopax.org

standards in computational biology, and provide a more lightweight "official" stamp of approval for standards than the quite involved processes of the IEEE, etc.

NeuroML is increasingly formalising the processes used for developing the language, including the election of a ten member Scientific Committee at the 2011 NeuroML Development Workshop (section 3.4). As NeuroML v2.0 and LEMS move from proof of concept implementations to being stable enough to replace v1.x, formal specification documents will be required, along with elected editors.

More interactions within the COMBINE forum will also encourage greater awareness of NeuroML and computational neuroscience in general among the systems biology community. This will lead to mutually advantageous exchange of knowledge between the disciplines, not only in the biological domain but with respect to the computational and simulation techniques used.

7.3 Collaborative large scale model development

7.3.1 Large scale networks

Distributed computing hardware is increasingly being used to create large scale network models with close to biological cell densities (Djurfeldt et al, 2008; Izhikevich and Edelman, 2008; Markram, 2006). Convincing experimental neuroscientists that these are more than exercises in pushing the boundaries of hardware and software, but are valid research tools, will require sharing of technical implementation details and source code, critical analysis of models and crucially, independent reproduction of simulation results.

Parallel NEURON and NEST are the leading freely available platforms for performing such simulations, the former concentrating on networks with more biophysically detailed cell elements, the latter on larger scale networks of simpler cells. neuroConstruct currently provides a framework for generating and managing large scale simulations on high performance computing hardware, both locally and on remote shared resources (section 4.2.3). This is currently focussed on Parallel NEURON, but much of the functionality is generic enough to be reused for other parallel simulators (with MOOSE being a prime candidate). Simulations of networks of 10^4 detailed cells and 10^6 simple cells are possible (section 4.4). Currently there is basic interaction with NEST through the generation of PyNN based simulations of Integrate and Fire neurons, but greater interaction will be possible with NeuroML v2.0.

In addition to usage within the Silver Lab and on the UCL high performance computing infrastructure (Table 4.1), large scale simulations generated by neuroConstruct are being tested in a number of other labs around the world. Simulations generated on a laptop in London have been sent to and executed on the Italian Supercomputing Consortium resource CASPUR in Rome as part of an ongoing collaboration with researchers in University of Pavia. The National Center for Microscopy and Imaging Research at UC San Diego, as part of the Whole Brain Catalog[9] project are using neuroConstruct to generate parallel network simulations of detailed network models constructed through their 3D graphical user interface. As part of the EPSRC COLAMN Project[10] neuroConstruct has been used to generate details network models based on their experimental work reconstructing the laminar microcircuitry of the neocortex. Researchers are also using neuroCon-

[9]http://www.wholebraincatalog.org
[10]http://colamn.plymouth.ac.uk/colamn-project

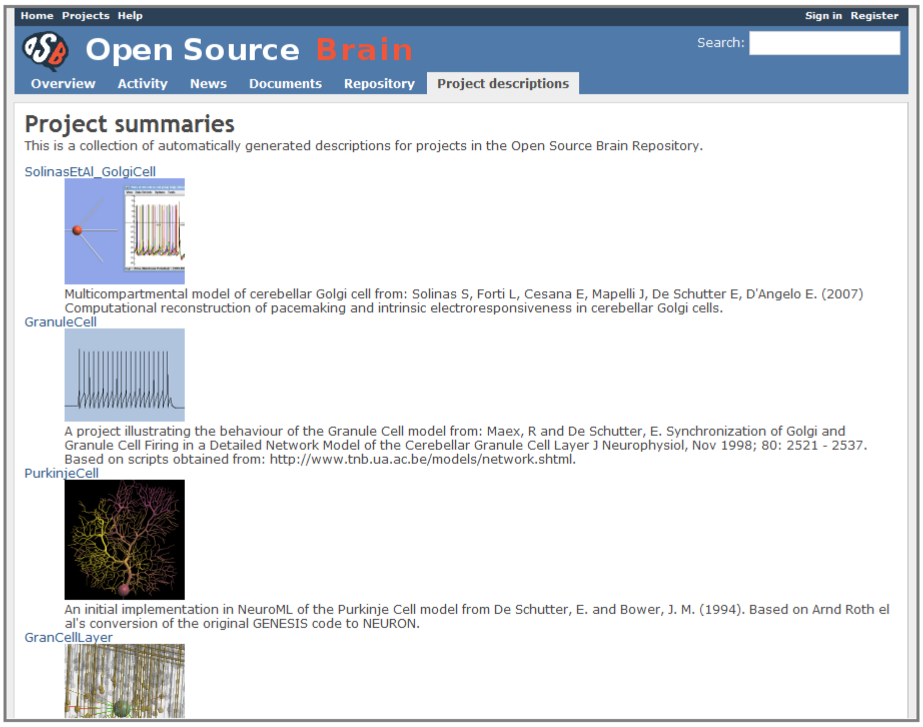
struct (and are planning to use the parallel code generation functionality) in Poland (Daniel Wójcik, thalamocortical networks), Hungary (Szabolcs Káli, hippocampus) and UC Irvine (Ivan Soltesz, dentate gyrus and hippocampus). Feedback from all of these groups will help improve the usability of this feature, and neuroConstruct in general, benefiting all users.

7.3.2 The Open Source Brain Initiative

The process of converting multiple cell and network models to neuroConstruct and NeuroML format (chapter 5) has been quite informative of how models develop over time. Model components are often reused between modelling projects, either channel or synaptic mechanisms, or whole cells. Tracking the provenance of these and seeing when they are reused (and/or updated) can be difficult. Bugs and outdated modelling practices are frequently found in files and correcting these can often change the behaviour of the model slightly. Keeping a record of any changes, who made them and more importantly why, and providing the ability to get back to a version of the model corresponding to an important revision in the past will require an infrastructure not present in current model repositories which focus on disseminating simulator specific models as used in a published article.

These issues of version control have been well addressed by the software development community. Packages such as CVS (Concurrent Versions System), Subversion (SVN) and Mercurial have been used for years by teams of developers to collaboratively coordinate software development projects[11]. Each revision is recorded with comments, projects can have development and stable branches and significant releases are tagged, giving users the choice of what version of the code to choose.

[11]Fro example for open source projects at http://SourceForge.net



Project summaries

This is a collection of automatically generated descriptions for projects in the Open Source Brain Repository.

SolinasEtAl_GolgiCell

Multicompartmental model of cerebellar Golgi cell from: Solinas S, Forti L, Cesana E, Mapelli J, De Schutter E, D'Angelo E. (2007) Computational reconstruction of pacemaking and intrinsic electroresponsiveness in cerebellar Golgi cells.

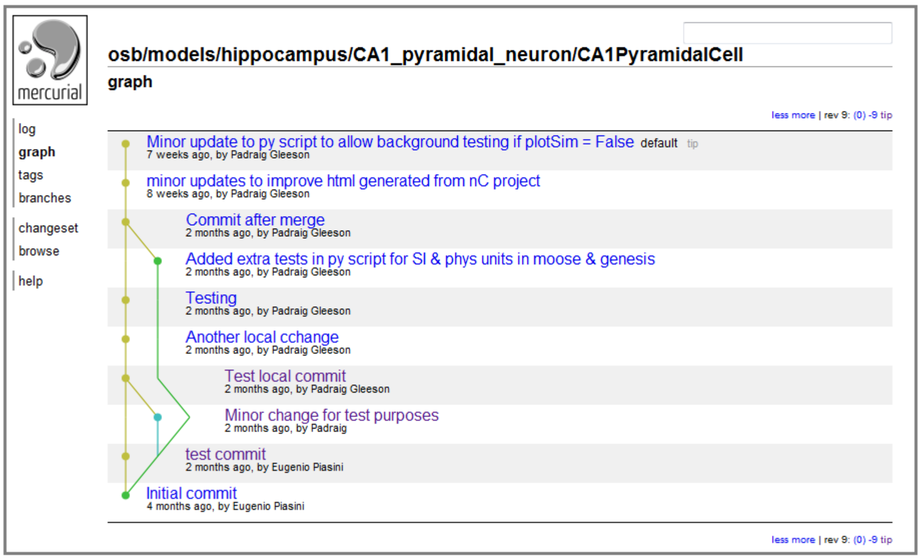
GranuleCell

A project illustrating the behaviour of the Granule Cell model from: Maex, R and De Schutter, E. Synchronization of Golgi and Granule Cell Firing in a Detailed Network Model of the Cerebellar Granule Cell Layer J Neurophysiol, Nov 1998; 80: 2521 - 2537. Based on scripts obtained from: http://www.tnb.ua.ac.be/models/network.shtml.

PurkinjeCell

An initial implementation in NeuroML of the Purkinje Cell model from De Schutter, E. and Bower, J. M. (1994). Based on Arnd Roth el al's conversion of the original GENESIS code to NEURON.

GranCellLayer



osb/models/hippocampus/CA1_pyramidal_neuron/CA1PyramidalCell

graph

Minor update to py script to allow background testing if plotSim = False default tip
7 weeks ago, by Padraig Gleeson

minor updates to improve html generated from nC project
8 weeks ago, by Padraig Gleeson

Commit after merge
2 months ago, by Padraig Gleeson

Added extra tests in py script for SI & phys units in moose & genesis
2 months ago, by Padraig Gleeson

Testing
2 months ago, by Padraig Gleeson

Another local cchange
2 months ago, by Padraig Gleeson

Test local commit
2 months ago, by Padraig Gleeson

Minor change for test purposes
2 months ago, by Padraig

test commit
2 months ago, by Eugenio Piasini

Initial commit
4 months ago, by Eugenio Piasini

Figure 7.3: Open Source Brain website. Top: main interface (available at http://OpenSourceBrain.org:8080) showing a list of some of the projects already available on the site. Bottom: view of revision graph for single project (CA1 pyramidal cell model). The history of changes for each project are recorded, including the developments which happen in distributed branches which are merged back into the main repository.

To utilise these modern software development practices and to encourage more collaborative model development among the computational neuroscience community, we have started the Open Source Brain initiative, to create an open repository of detailed neuronal models of widespread interest to the neuroscience community. This will consist of a Mercurial based version control repository containing the models (the set of models described in chapter 5 comprise the initial contents), a web front end to provide readable summaries of model contents, and an integrated set of tools which interact with the repository. An initial implementation of this repository has been made publicly available[12], and screenshots are shown in Figure 7.3.

Our work on the repository in the short term will focus on improving the cortical and cerebellar models in the repository as these are of particular interest within the Silver Lab. This will involve collaboration with identified partners who are interested in making their models more accessible, and getting access to more stable and well tested cell models for their research. Linking simulations to experimental data (either stored alongside the models in the repository, or preferably in a dedicated external database) will also be crucial for demonstrating the biological relevance of the data. Our work with these initial collaborators will help resolve this and other practical considerations of the underlying infrastructure and improve the usability of the repository as we expand and encourage the wider community to get involved.

[12]At http://OpenSourceBrain.org:8080, Euginio Piasini has assisted greatly in setting up this web frontend

7.4 Future plans

A stable specification for NeuroML version 2.0, greater support for this in neuroConstruct and an expanding core set of cell and network models in the Open Source Brain repository will be at the heart of my future efforts in this area, and are in line with the ongoing objectives of this work (chapter 6). Increased opportunities for creating both large scale abstract neuronal networks and truly multiscale detailed models, all expressed in a common language, will be a real driver for new and interesting tools in this area. A growing repository of stable, well tested and biologically relevant neuronal models together with a wide range of applications that can read them can only encourage neuroscience from non computational backgrounds and even non neuroscientists to use these resources as a part of their research, addressing questions of neuronal function from diverse points of view.

The technical challenges in the near future include better integration of NeuroML and neuroConstruct with PyNN and therefore NEST to allow greater interaction with researchers interested in large scale networks of simplified neurons. This will also lead to better interaction with the range of other (mainly Python based) data analysis and simulation tools being developed through the NeuralEnsemble website[13].

Interoperability with SBML and CellML will also be expanded. The wealth of biologically relevant, standardised models in the BioModels database and CellML Model Repository will be crucial for developing neuronal models incorporating many levels of biological scale. There are also important initiatives happening in Japan including the Physiome.jp project (developing the InSilicoML language (Nomura, 2010)) to develop large scale *in silico* models of human physiology, and the various initiatives being developed by

[13]http://neuralensemble.org

the INCF Japan Node[14]. COMBINE (section 7.2.3) will provide a key forum for greater collaboration and discussion between all of these different initiatives. The INCF will also provide a forum for collaboration. In addition to the Program on Multiscale Modelling, the Digital Brain Atlasing, Ontologies of Neural Structures and Standards for Datasharing Programs are all active in the development of interoperable software, and standards for annotating and sharing data.

Updates to neuroConstruct will involve using NeuroML v2.0 at a very low level for the description of cellular components and networks. This will allow even more portability of scripts for interacting with neuroConstruct (e.g. to access and change cell or network parameters in an optimisation routine) across other NeuroML compliant software. Other extensions planned include a new 3D visualisation engine based on dedicated high performance graphics libraries[15], further increasing a users ability to display and interact with large scale neuronal models.

Intriguing possibilities are being investigated to facilitate setting up and accessing dedicated neuronal simulation compute nodes which will run the large scale models generated by neuroConstruct. Neurodebian[16] is an initiative to simplify setup of neuroscience related software in many popular Linux distributions. This facilitates creation of new servers (or virtual machines) containing compatible, stable versions of neuroscience applications. I plan to integrate neuroConstruct into this framework, offering the opportunity to users to get the application, along with (Parallel) NEURON, NEST, PyNN or MOOSE, installed and configured in minutes with minimal user interaction. This initiative will also facilitate computational neuroscience

[14]http://www.neuroinf.jp/platforms
[15]For example jMonkeyEngine (http://jmonkeyengine.org), a Java based gaming engine
[16]http://neuro.debian.net

ready servers being available in “the Cloud”[17]. Use of on demand computational resources is increasing, with companies like Amazon, Google and Apple offering storage and computing resources on a pay as you go basis. The possibility of creating a simulation in neuroConstruct, asking for 100 Neurodebian configured nodes from one of these providers and executing the simulation across them, is an exciting and potentially very cost effective alternative to each lab purchasing and maintaining their own dedicated clusters.

Active collaboration on improving the cerebellar and cortical models in the Open Source Brain repository will continue, with both specific research questions in mind, and with a view to providing a suite of test models for improving associated simulation and analysis tools. Expansion of the range of models in the repository to other brain regions will also be actively encouraged, with the hippocampus, basal ganglia and some higher level systems models (tadpole, C. elegans) as key targets in the medium term. With the online collaboration infrastructure improving based on experiences with the initial core set of models and collaborators, we will be able to offer an an attractive environment to researchers to share, promote and improve their models.

The challenge of understanding the brain in normal and pathological states is one which will require a wide range of experimental and theoretical approaches. Computational modelling holds great promise for the future to consolidate this knowledge and provide new insights. It is hoped that the range of tools, languages and models presented here will help move the field forward, contribute to a solid basis for the software infrastructure which will be required and make it easier for researchers to create, simulate,

[17]See http://software.incf.org/software/incf-cloud-app

analyse and critically assess each other's models, improving our collective understanding of how the nervous system functions.

Bibliography

Agmon-Snir H, Carr CE, Rinzel J (1998) The role of dendrites in auditory coincidence detection. Nature 393(6682):268–72

Albus JS (1971) A theory of cerebellar function. Mathematical Biosciences 10(1-2):25–61

Apps R, Garwicz M (2005) Anatomical and physiological foundations of cerebellar information processing. Nature Reviews Neuroscience 6(4):297–311

Ascoli GA (2006) Mobilizing the base of neuroscience data: the case of neuronal morphologies. Nature Reviews Neuroscience 7(4):318–24

Ascoli GA, Donohue DE, Halavi M (2007) NeuroMorpho.org: a central resource for neuronal morphologies. Journal of Neuroscience 27(35):9247–51

Basalyga G, Gleiser PM, Wennekers T (2011) Emergence of small-world structure in networks of spiking neurons through STDP plasticity. In: Hernandez C, Sanz R, Gomez-Ramirez J, Smith L, Hussain A, Chella A, Aleksander I (eds) From Brains to Systems, Advances in Experimental Medicine and Biology, Springer New York

Bean BP (2007) The action potential in mammalian central neurons. Nature Reviews Neuroscience 8(6):451–65

Bednar JA (2009) Topographica: Building and analyzing map-level simulations from Python, C/C++, MATLAB, NEST, or NEURON components. Frontiers in Neuroinformatics 3:8

Bell CC, Han V, Sawtell NB (2008) Cerebellum-like structures and their implications for cerebellar function. Annual Review of Neuroscience 31(1):1–24

Berends M, Maex R, Schutter ED (2004) A detailed three-dimensional model of the cerebellar granular layer. Neurocomputing 58-60:587–592

Berends M, Maex R, Schutter ED (2005) The effect of NMDA receptors on gain modulation. Neural Computationation 17(12):2531–2547

Blackwell KT (2006) An efficient stochastic diffusion algorithm for modeling second messengers in dendrites and spines. Journal of Neuroscience Methods 157(1):142–153

Bower J, Beeman D (1997) The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System. Springer, New York

Boyden ES, Zhang F, Bamberg E, Nagel G, Deisseroth K (2005) Millisecond-timescale, genetically targeted optical control of neural activity. Nature Neuroscience 8(9):1263–8

Braitenberg V, Atwood RP (1958) Morphological observations on the cerebellar cortex. The Journal of Comparative Neurology 109(1):1–33

Bray T, Paoli J, Sperberg-McQueen CM (1998) Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/REC-xml

Brette R, Gerstner W (2005) Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. Journal of Neurophysiology 94(5):3637–3642

Brette R, Rudolph M, Carnevale T, Hines M, Beeman D, Bower JM, Diesmann M, Morrison A, Goodman PH, Harris J F C, Zirpe M, Natschlager T, Pecevski D, Ermentrout B, Djurfeldt M, Lansner A, Rochel O, Vieville T, Muller E, Davison AP, El Boustani S, Destexhe A (2007) Simulation of networks of spiking neurons: a review of tools and strategies. Journal of Computational Neuroscience 23(3):349–98

Briggman KL, Denk W (2006) Towards neural circuit reconstruction with volume electron microscopy techniques. Current Opinion in Neurobiology 16(5):562–570

Brunel N, van Rossum M (2007) Lapicque's 1907 paper: From frogs to integrate-and-fire. Biological Cybernetics 97:337–339

Cannon R, Gewaltig MO, Gleeson P, Bhalla U, Cornelis H, Hines M, Howell F, Muller E, Stiles J, Wils S, De Schutter E (2007) Interoperability of neuroscience modeling software: Current status and future directions. Neuroinformatics 5(2):127–138

Cannon RC, Turner DA, Pyapali GK, Wheal HV (1998) An on-line archive of reconstructed hippocampal neurons. Journal of Neuroscience Methods 84(1-2):49–54

Cannon RC, O'Donnell C, Nolan MF (2010) Stochastic ion channel gating in dendritic neurons: Morphology dependence and probabilistic synaptic activation of dendritic spikes. PLoS Computational Biology 6(8):e1000,886

Carnevale NT, Hines ML (2006) The NEURON Book. Cambridge University Press

Chemla S, Chavane F (2010) A biophysical cortical column model to study the multi-component origin of the VSDI signal. NeuroImage 53(2):420–438

Coggan JS, Bartol TM, Esquenazi E, Stiles JR, Lamont S, Martone ME, Berg DK, Ellisman MH, Sejnowski TJ (2005) Evidence for ectopic neurotransmission at a neuronal synapse. Science 309(5733):446–451

Cohen D, Yarom Y (1998) Patches of synchronized activity in the cerebellar cortex evoked by mossy-fiber stimulation: Questioning the role of parallel fibers. Proceedings of the National Academy of Sciences 95(25):15,032–15,036

Cornelis H, De Schutter E (2003) NeuroSpaces: separating modeling and simulation. Neurocomputing 52(4):227–231

Crook S, Gleeson P, Howell F, Svitak J, Silver RA (2007) MorphML: Level 1 of the NeuroML standards for neuronal morphology data and model specification. Neuroinformatics 5(2):96–104

Cunningham MO, Whittington MA, Bibbig A, Roopun A, LeBeau FEN, Vogt A, Monyer H, Buhl EH, Traub RD (2004) A role for fast rhythmic bursting neurons in cortical gamma oscillations in vitro. Proceedings of the National Academy of Sciences 101(18):7152–7157

Cuntz H, Forstner F, Borst A, Häusser M (2010) One rule to grow them all: A general theory of neuronal branching and its practical application. PLoS Computational Biology 6(8):e1000,877

Cutsuridis V, Graham B, Cobb S, Vida I (eds) (2010) Hippocampal Microcircuits: A Computational Modeler's Resource Book. Springer Series in Computational Neuroscience, Springer New York

D'Angelo E, Mazzarello P, Prestori F, Mapelli J, Solinas S, Lombardo P, Cesana E, Gandolfi D, Congi L (2011) The cerebellar network: From structure to function and dynamics. Brain Research Reviews 66(1-2):5–15

Davison AP, Bruderle D, Eppler J, Kremkow J, Muller E, Pecevski D, Perrinet L, Yger P (2008) PyNN: A common interface for neuronal network simulators. Frontiers in Neuroinformatics 2:11

Davison AP, Hines ML, Muller E (2009) Trends in programming languages for neuroscience simulations. Frontiers in Neuroscience 3(3):374–80

Dayan P (2006) Levels of analysis in neural modeling. In: Encyclopedia of Cognitive Science, John Wiley and Sons, Ltd

De Schutter E (2008) Why are computational neuroscience and systems biology so separate? PLoS Computational Biology 4(5):e1000,078

De Schutter E, Bower JM (1994) An active membrane model of the cerebellar Purkinje cell. I. simulation of current clamps in slice. Journal of Neurophysiology 71(1):375–400

Destexhe A (2009) Self-sustained asynchronous irregular states and up-down states in thalamic, cortical and thalamocortical networks of non-linear integrate-and-fire neurons. Journal of Computational Neuroscience 27:493–506

Destexhe A, Huguenard JR (2000) Which formalism to use for modeling voltage-dependent conductances? In: Schutter ED (ed) Computational

Neuroscience: Realistic Modeling for Experimentalists, CRC Press, Boca Raton FL

Destexhe A, Pare D (1999) Impact of network activity on the integrative properties of neocortical pyramidal neurons in vivo. Journal of Neurophysiology 81(4):1531–47

Diesmann M, Gewaltig MO (2002) NEST: An Environment for Neural Systems Simulations, vol Forschung und wisschenschaftliches Rechnen, Beitrage zum Heinz-Billing-Preis 2001. Gottingen: Ges. fur Wiss. Datenverarbeitung

Dino MR, Schuerger RJ, Liu YB, Slater NT, Mugnaini E (2000) Unipolar brush cell: a potential feedforward excitatory interneuron of the cerebellum. Neuroscience 98(4):625–636

Diwakar S, Lombardo P, Solinas S, Naldi G, D'Angelo E (2011) Local field potential modeling predicts dense activation in cerebellar granule cells clusters under LTP and LTD control. PLoS ONE 6(7):e21,928

Djurfeldt M, Lansner A (2007) Workshop report: 1st INCF workshop on large-scale modeling of the nervous system. Nature Precedings (http://dx.doi.org/10.1038/npre.2007.262.1)

Djurfeldt M, Lundqvist M, Johansson C, Rehn M, Ekeberg O, Lansner A (2008) Brain-scale simulation of the neocortex on the IBM Blue Gene/L supercomputer. IBM Journal of Research and Development 52(1.2):31–41

Douglas RJ, Martin KA (2007) Mapping the matrix: The ways of neocortex. Neuron 56(2):226–238

Douglas RJ, Martin KAC (2004) Neuronal circuits of the neocortex. Annual Review of Neuroscience 27(1):419–451

Drewes R, Zou Q, Goodman PH (2009) Brainlab: A Python toolkit to aid in the design, simulation, and analysis of spiking neural networks with the NeoCortical Simulator. Frontiers in Neuroinformatics 3:16

Druckmann S, Banitt Y, Gidon AA, Schürmann F, Markram H, Segev I (2007) A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data. Frontiers in Neuroscience 1(0)

Dugué GP, Brunel N, Hakim V, Schwartz E, Chat M, Lévesque M, Courtemanche R, Léna C, Dieudonné S (2009) Electrical coupling mediates tunable low-frequency oscillations and resonance in the cerebellar Golgi cell network. Neuron 61(1):126–139

Eccles JC, Ito M, Szentágothai J (1967) The Cerebellum as a Neuronal Machine. Springer-Verlag, New York

Eppler JM, Helias M, Muller E, Diesmann M, Gewaltig MO (2008) PyNEST: A convenient interface to the NEST simulator. Frontiers in Neuroinformatics 2:12

Ermentrout B (2002) Simulating, Analyzing, and Animating Dynamical Systems: A Guide To XPPAUT for Researchers and Students. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA

Farinella M, Gleeson P, Ruedt D, Silver A (2011) Synaptic integration and NMDA spikes in a layer 5 pyramidal neuron model. In: Proceedings of the Twentieth Annual Computational Neuroscience Meeting, CNS*2011

Fidjeland AK, Shanahan MP (2010) Accelerated simulation of spiking neural networks using GPUs. In: Proc. IEEE International Joint Conference on Neural Networks

Fink M, Noble D (2009) Markov models for ion channels: versatility versus identifiability and speed. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 367(1896):2161–2179

FitzHugh R (1961) Impulses and physiological states in theoretical models of nerve membrane. Biophysical Journal 1(6):445–466

Gabbiani F, Midtgaard J, Knopfel T (1994) Synaptic integration in a model of cerebellar granule cells. Journal of Neurophysiology 72(2):999–1009

Gardner D, Knuth KH, Abato M, Erde SM, White T, DeBellis R, Gardner EP (2001) Common data model for neuroscience data and data model exchange. Journal of the American Medical Informatics Association 8(1):17–33

Gell-Mann M (1995) The Quark and the Jaguar: Adventures in the Simple and the Complex. Abacus

Gerstner W, Kistler W (2002) Spiking neuron models: single neurons, populations, plasticity. Cambridge University Press

Gleeson P, Steuber V, Silver RA (2007) neuroConstruct: a tool for modeling networks of neurons in 3D space. Neuron 54(2):219–35

Gleeson P, Steuber V, Silver RA (2008) Using neuroConstruct to develop and modify biologically detailed 3D neuronal network models in health and disease. In: Soltesz I, Staley K (eds) Computational Neuroscience in Epilepsy, Academic Press, San Diego, pp 48–70

Gleeson P, Crook S, Cannon RC, Hines ML, Billings GO, Farinella M, Morse TM, Davison AP, Ray S, Bhalla US, Barnes SR, Dimitrova YD, Silver RA (2010a) NeuroML: A language for describing data driven models of neurons and networks with a high degree of biological detail. PLoS Computational Biology 6(6):e1000,815

Gleeson P, Silver RA, Steuber V (2010b) Computer Simulation Environments. In: Cutsuridis V, Graham B, Cobb S, Vida I (eds) Hippocampal Microcircuits, Springer Series in Computational Neuroscience, vol 5, Springer New York, pp 593–609

Gleeson P, Steuber V, Silver RA, Crook S (in press) NeuroML. In: Le Novère N, Bhalla US (eds) Computational Systems Neurobiology, Springer

Goddard NH, Hucka M, Howell F, Cornelis H, Shankar K, Beeman D (2001) Towards NeuroML: model description methods for collaborative modelling in neuroscience. Philos Trans R Soc Lond B Biol Sci 356(1412):1209–28

Golowasch J, Goldman MS, Abbott LF, Marder E (2002) Failure of averaging in the construction of a conductance-based neuron model. Journal of Neurophysiology 87(2):1129–1131

Goodman D (2010) Code generation: A strategy for neural network simulators. Neuroinformatics 8:183–196

Goodman D, Brette R (2008) Brian: A simulator for spiking neural networks in Python. Frontiers in Neuroinformatics 2:5

Gouwens NW, Wilson RI (2009) Signal propagation in drosophila central neurons. The Journal of Neuroscience 29(19):6239–6249

Graham LJ (2002) Modelling neuronal biophysics. In: Arbib M (ed) Handbook of Brain Theory and Neural Networks, 2nd edn, MIT Press, Cambridge, Massachusetts

Grillner S, Markram H, Schutter ED, Silberberg G, LeBeau FE (2005) Microcircuits in action: from CPGs to neocortex. Trends in Neurosciences 28(10):525–533

Gurney K, Humphries M (in press) Methodological issues in modelling at multiple levels of description. In: Le Novère N, Bhalla US (eds) Computational Systems Neurobiology, Springer

Hansel C, Linden DJ, D'Angelo E (2001) Beyond parallel fiber LTD: the diversity of synaptic and non-synaptic plasticity in the cerebellum. Nature Neuroscience 4(5):467–75

Hanson J, Madison D (2010) Imbalanced pattern completion vs. separation in cognitive disease: network simulations of synaptic pathologies predict a personalized therapeutics strategy. BMC Neuroscience 11(1):96

Harvey R, Napper R (1991) Quantitatives studies on the mammalian cerebellum. Progress in Neurobiology 36(6):437–463

Harvey RJ, Napper RMA (1988) Quantitative study of granule and Purkinje cells in the cerebellar cortex of the rat. The Journal of Comparative Neurology 274(2):151–157

Helmstaedter M, de Kock C, Feldmeyer D, Bruno R, Sakmann B (2007) Reconstruction of an average cortical column in silico. Brain Research Reviews 55(2):193–203

Herz AVM, Gollisch T, Machens CK, Jaeger D (2006) Modeling single-neuron dynamics and computations: A balance of detail and abstraction. Science 314(5796):80–85

Hille B (2001) Ionic channels of excitable membranes, 3rd edn. Sinauer Associates, Sunderland, Mass.

Hines M (1984) Efficient computation of branched nerve equations. International Journal of Bio-Medical Computing 15(1):69–76

Hines ML, Carnevale NT (2000) Expanding NEURON's repertoire of mechanisms with NMODL. Neural Computationation 12(5):995–1007

Hines ML, Carnevale NT (2008) Translating network models to parallel hardware in NEURON. Journal of Neuroscience Methods 169(2):425–55

Hines ML, Morse T, Migliore M, Carnevale NT, Shepherd GM (2004) ModelDB: A database to support computational neuroscience. Journal of Computational Neuroscience 17(1):7–11

Hines ML, Eichner H, Schürmann F (2008a) Neuron splitting in compute-bound parallel network simulations enables runtime scaling with twice as many processors. Journal of Computational Neuroscience 25(1):203–10

Hines ML, Eichner H, Schürmann F (2008b) Neuron splitting in compute-bound parallel network simulations enables runtime scaling with twice as many processors. Journal of Computational Neuroscience 25(1):203–10

Hines ML, Markram H, Schürmann F (2008c) Fully implicit parallel simulation of single neurons. Journal of Computational Neuroscience 25(3):439–48

Hines ML, Davison AP, Muller E (2009) NEURON and Python. Frontiers in Neuroinformatics 3:1

Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. The Journal of Physiology 117(4):500–544

Holmes G (1917) The symptoms of acute cerebellar injuries due to gunshot injuries. Brain 40(4):461–535

Howell F, Cannon R, Goddard N, Bringmann H, Rogister P, Cornelis H (2003) Linking computational neuroscience simulation tools–a pragmatic approach to component-based development. Neurocomputing 52-54:289–294

Howell FW, Dyhrfjeld-Johnsen J, Maex R, Goddard N, Schutter ED (2000) A large-scale model of the cerebellar cortex using PGENESIS. Neurocomputing 32-33:1041–1046

Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, Arkin AP, Bornstein BJ, Bray D, Cornish-Bowden A, Cuellar AA, Dronov S, Gilles ED, Ginkel M, Gor V, Goryanin I, Hedley WJ, Hodgman TC, Hofmeyr JH, Hunter PJ, Juty NS, Kasberger JL, Kremling A, Kummer U, Le Novère N, Loew LM, Lucio D, Mendes P, Minch E, Mjolsness ED, Nakayama Y, Nelson MR, Nielsen PF, Sakurada T, Schaff JC, Shapiro BE, Shimizu TS, Spence HD, Stelling J, Takahashi K, Tomita M, Wagner J, Wang J (2003) The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19(4):524–31

Ito M, Sakurai M, Tongroach P (1982) Climbing fibre induced depression of both mossy fibre responsiveness and glutamate sensitivity of cerebellar Purkinje cells. The Journal of Physiology 324(1):113–134

Izhikevich E (2003) Simple model of spiking neurons. Neural Networks, IEEE Transactions on 14(6):1569–1572

Izhikevich EM, Edelman GM (2008) Large-scale model of mammalian thalamocortical systems. Proceedings of the National Academy of Sciences 105(9):3593–3598

Jarsky T, Roxin A, Kath WL, Spruston N (2005) Conditional dendritic spike propagation following distal synaptic activation of hippocampal CA1 pyramidal neurons. Nature Neuroscience 8(12):1667–76

Johnston D, Narayanan R (2008) Active dendrites: colorful wings of the mysterious butterflies. Trends in Neurosciences 31(6):309–316

Kanichay RT, Silver RA (2008) Synaptic and cellular properties of the feedforward inhibitory circuit within the input layer of the cerebellar cortex. The Journal of Neuroscience 28(36):8955–8967

Kirkby PA, Nadella KMNS, Silver RA (2010) A compact acousto-optic lens for 2D and 3D femtosecond based 2-photon microscopy. Optics Express 18(13):13,720–13,744

Kitano H (2002) Systems biology: A brief overview. Science 295(5560):1662–1664

Klausberger T, Somogyi P (2008) Neuronal diversity and temporal dynamics: The unity of hippocampal circuit operations. Science 321(5885):53–57

Ko H, Hofer SB, Pichler B, Buchanan KA, Sjöström PJ, Mrsic-Flogel TD (2011) Functional specificity of local synaptic connections in neocortical networks. Nature 473(7345):87–91

Koch C, Segev I (2000) The role of single neurons in information processing. Nature Neuroscience 3 Suppl:1171–7

Koene R, Tijms B, van Hees P, Postma F, de Ridder A, Ramakers G, van Pelt J, van Ooyen A (2009) NETMORPH: A framework for the stochastic generation of large scale neuronal networks with realistic neuron morphologies. Neuroinformatics 7:195–210

Kohl P, Noble D (2009) Systems Biology and the Virtual Physiological Human. Molecular Systems Biology 5(292):1–6

Kole MHP, Ilschner SU, Kampa BM, Williams SR, Ruben PC, Stuart GJ (2008) Action potential generation requires a high sodium channel density in the axon initial segment. Nature Neuroscience 11(2):178–186

Kopell N (2005) Does it have to be this complicated? Focus on “Single-column thalamocortical network model exhibiting gamma oscillations, spindles, and epileptogenic bursts”. Journal of Neurophysiology 93(4):1829–1830, DOI 10.1152/jn.01147.2004

Kotaleski JH, Blackwell KT (2010) Modelling the molecular mechanisms of synaptic plasticity using systems biology approaches. Nature Reviews Neuroscience 11(4):239–51

Kumar A, Rotter S, Aertsen A (2008) Conditions for propagating synchronous spiking and asynchronous firing rates in a cortical network model. The Journal of Neuroscience 28(20):5268–5280

Lansner A, Diesmann M (in press) Virtues, pitfalls, and methodology related to large-scale neuronal network models and supercomputer simulations. In: Le Novère N, Bhalla US (eds) Computational Systems Neurobiology, Springer

Lavrentovich M, Hemkin S (2008) A mathematical model of spontaneous calcium(II) oscillations in astrocytes. Journal of Theoretical Biology 251(4):553–560

Le Novère N, Bornstein B, Broicher A, Courtot M, Donizelli M, Dharuri H, Li L, Sauro H, Schilstra M, Shapiro B, Snoep JL, Hucka M (2006) BioModels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. Nucleic Acids Res 34(Database issue):D689–91

Lefort S, Tomm C, Sarria JCF, Petersen CC (2009) The excitatory neuronal network of the C2 barrel column in mouse primary somatosensory cortex. Neuron 61(2):301–316

Li C, Donizelli M, Rodriguez N, Dharuri H, Endler L, Chelliah V, Li L, He E, Henry A, Stefan MI, Snoep JL, Hucka M, Le Novère N, Laibe C (2010) BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models. BMC Systems Biology 4:92

Lloyd CM, Halstead MD, Nielsen PF (2004) CellML: its future, present and past. Progress in Biophysical Mololecular Bioliology 85(2-3):433–50

Lloyd CM, Lawson JR, Hunter PJ, Nielsen PF (2008) The CellML Model Repository. Bioinformatics 24(18):2122–3

London M, Häusser M (2005) Dendritic computation. Annual Review of Neuroscience 28(1):503–532

Lübke J, Feldmeyer D (2007) Excitatory signal flow and connectivity in a cortical column: focus on barrel cortex. Brain Structure and Function 212:3–17

Lübke J, Roth A, Feldmeyer D, Sakmann B (2003) Morphometric analysis of the columnar innervation domain of neurons connecting layer 4 and layer 2/3 of juvenile rat barrel cortex. Cerebral Cortex 13(10):1051–1063

Maex R, Schutter ED (1998) Synchronization of Golgi and granule cell firing in a detailed network model of the cerebellar granule cell layer. Journal of Neurophysiology 80(5):2521–2537

Mainen ZF, Sejnowski TJ (1996) Influence of dendritic structure on firing pattern in model neocortical neurons. Nature 382(6589):363–6

Mainen ZF, Joerges J, Huguenard JR, Sejnowski TJ (1995) A model of spike initiation in neocortical pyramidal neurons. Neuron 15(6):1427–39

Marder E, Taylor AL (2011) Multiple models to capture the variability in biological neurons and networks. Nature Neuroscience 14(2):133–8

Markram H (2006) The Blue Brain Project. Nature Reviews Neuroscience 7(2):153–160

Marr D (1969) A theory of cerebellar cortex. Journal of Physiology 202(2):437–470

Marr D (1982) Vision: a computational investigation into the human representation and processing of visual information. Freeman, New York

Marr D, Poggio T (1977) From understanding computation to understanding neural circuitry. Neurosciences Research Program Bulletin 15:470–488

McCulloch W, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biology 5:115–133

Medina JF, Garcia KS, Nores WL, Taylor NM, Mauk MD (2000) Timing mechanisms in the cerebellum: Testing predictions of a large-scale computer simulation. The Journal of Neuroscience 20(14):5516–5525

Migliore M, Shepherd GM (2002) Emerging rules for the distributions of active dendritic conductances. Nature Reviews Neuroscience 3(5):362–70

Migliore M, Cook EP, Jaffe DB, Turner DA, Johnston D (1995) Computer simulations of morphologically reconstructed CA3 hippocampal neurons. Journal of Neurophysiology 73(3):1157–68

Migliore M, Ferrante M, Ascoli GA (2005) Signal propagation in oblique dendrites of CA1 pyramidal cells. Journal of Neurophysiology 94(6):4145–4155

Migliore M, Cannia C, Lytton W, Markram H, Hines M (2006) Parallel network simulations with NEURON. Journal of Computational Neuroscience 12(2):119–129

Molineux ML, Fernandez FR, Mehaffey WH, Turner RW (2005) A-type and T-type currents interact to produce a novel spike latency-voltage relationship in cerebellar stellate cells. The Journal of Neuroscience 25(47):10,863–10,873

Morris C, Lecar H (1981) Voltage oscillations in the barnacle giant muscle fiber. Biophysical Journal 35(1):193–213

Morrison A, Mehring C, Geisel T, Aertsen AD, Diesmann M (2005) Advancing the boundaries of high-connectivity network simulation with distributed computing. Neural Computation 17(8):1776–801

Napper RMA, Harvey RJ (1988) Number of parallel fiber synapses on an individual Purkinje cell in the cerebellum of the rat. The Journal of Comparative Neurology 274(2):168–177

Nieus T, Sola E, Mapelli J, Saftenku E, Rossi P, D'Angelo E (2006) LTP regulates burst initiation and frequency at mossy fiber-granule cell synapses of rat cerebellum: Experimental observations and theoretical predictions. Journal of Neurophysiology 95(2):686–699

Noble D (2002) Modeling the heart: from genes to cells to the whole organ. Science 295(5560):1678–1682

Nomura T (2010) Towards integration of biological and physiological functions at multiple levels. Frontiers in Physiology 1(0)

Nordlie E, Gewaltig MO, Plesser HE (2009) Towards reproducible descriptions of neuronal network models. PLoS Computational Biology 5(8):e1000,456

O'Connor D, Huber D, Svoboda K (2009) Reverse engineering the mouse brain. Nature 461(7266):923–929

Ohyama T, Medina JF, Nores WL, Mauk MD (2002) Trying to understand the cerebellum well enough to build one. Annals of the New York Academy of Sciences 978(1):425–438

van Ooyen A (2011) Using theoretical models to analyse neural development. Nature Reviews Neuroscience 12(6):311–26

van Ooyen A, Duijnhouwer J, Remme MWH, van Pelt J (2002) The effect of dendritic topology on firing patterns in model neurons. Network: Computation in Neural Systems 13(3):311–325

Palay SL, Chan-Palay V (1974) Cerebellar Cortex: Cytology and Organization. Springer-Verlag, New York

Pecevski D, Natschlger T, Schuch K (2009) PCSIM: A parallel simulation environment for neural circuits fully integrated with Python. Frontiers in Neuroinformatics 3:11

Petersen CC (2007) The functional organization of the barrel cortex. Neuron 56(2):339–355

Plesser H, Eppler J, Morrison A, Diesmann M, Gewaltig MO (2007) Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. In: Kermarrec AM, Boug L, Priol T (eds) Euro-Par 2007 Parallel Processing, Lecture Notes in Computer Science, vol 4641, Springer Berlin / Heidelberg, pp 672–681

Poirazi P, Brannon T, Mel BW (2003) Pyramidal neuron as two-layer neural network. Neuron 37(6):989–99

Prinz AA, Bucher D, Marder E (2004) Similar network activity from disparate circuit parameters. Nature Neuroscience 7(12):1345–52

Qi W, Crook S (2004) Tools for neuroinformatic data exchange: an XML application for neuronal morphology data. Neurocomputing 58-60:1091–1095

Rall W (1959) Branching dendritic trees and motoneuron membrane resistivity. Experimental Neurology 1(5):491–527

Rall W, Agmon-Snir H (1998) Cable theory for dendritic neurons. In: Koch C, Segev I (eds) Methods in Neuronal Modeling, MIT Press

Rancz EA, Franks KM, Schwarz MK, Pichler B, Schaefer AT, Margrie TW (2011) Transfection via whole-cell recording in vivo: bridging single-cell physiology, genetics and connectomics. Nat Neurosci 14(4):527–32

Ray S, Bhalla US (2008) PyMOOSE: interoperable scripting in Python for MOOSE. Frontiers in Neuroinformatics 2:6(2)

Rhodes PA, Gray CM (1994) Simulations of intrinsically bursting neocortical pyramidal neurons. Neural Computation 6:1086–1110

Rokni D, Llinas RR, Yarom Y (2008) The morpho/functional discrepancy in the cerebellar cortex: Looks alone are deceptive. Frontiers in Neuroscience 2(0)

Rothman JS, Cathala L, Steuber V, Silver RA (2009) Synaptic depression enables neuronal gain control. Nature 457(7232):1015–8

Santhakumar V, Aradi I, Soltesz I (2005) Role of mossy fiber sprouting and mossy cell loss in hyperexcitability: a network model of the dentate gyrus incorporating cell types and axonal topography. Journal of Neurophysiology 93(1):437–53

Schilstra MJ, Li L, Matthews J, Finney A, Hucka M, Le Novre N (2006) CellML2SBML: conversion of CellML into SBML. Bioinformatics 22(8):1018–1020

Segev I, Burke RE (1998) Compartmental models of complex neurons. In: Koch C, Segev I (eds) Methods in Neuronal Modeling, MIT Press

Segev I, London M (2000) Untangling dendrites with quantitative models. Science 290(5492):744–750

Seyfarth EA (2006) Julius Bernstein (1839-1917): pioneer neurobiologist and biophysicist. Biological Cybernetics 94:2–8

Shepherd G (2004) The synaptic organization of the brain. Oxford University Press

Silver RA (2010) Neuronal arithmetic. Nature Reviews Neuroscience 11(7):474–89

Silver RA, Lübke J, Sakmann B, Feldmeyer D (2003) High-probability uniquantal transmission at excitatory synapses in barrel cortex. Science 302(5652):1981–1984

Single S, Borst A (1998) Dendritic integration and its role in computing image velocity. Science 281(5384):1848–1850

Solinas S, Forti L, Cesana E, Mapelli J, De Schutter E, D'Angelo E (2007a) Computational reconstruction of pacemaking and intrinsic electroresponsiveness in cerebellar Golgi cells. Frontiers in Cellular Neuroscience 1:2

Solinas S, Forti L, Cesana E, Mapelli J, De Schutter E, D'Angelo E (2007b) Fast-reset of pacemaking and theta-frequency resonance patterns in cerebellar Golgi cells: simulations of their impact in vivo. Frontiers in Cellular Neuroscience 1:4

Solinas S, Nieus T, D'Angelo E (2010) A realistic large-scale model of the cerebellum granular layer predicts circuit spatio-temporal filtering properties. Frontiers in Cellular Neuroscience 4:12

Somogyi P, Tamas G, Lujan R, Buhl EH (1998) Salient features of synaptic organisation in the cerebral cortex. Brain Research Reviews 26(2-3):113–135

Song S, Miller KD, Abbott LF (2000) Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. Nature Neuroscience pp 919–926

Song S, Sjöström PJ, Reigl M, Nelson S, Chklovskii DB (2005) Highly nonrandom features of synaptic connectivity in local cortical circuits. PLoS Biology 3(3):e68

Sporns O, Kotter R (2004) Motifs in brain networks. PLoS Biology 2(11):e369

Stanley D, Bardakjian B, Spano M, Ditto W (2011) Stochastic amplification of calcium-activated potassium currents in Ca^{2+} microdomains. Journal of Computational Neuroscience pp 1–20

Steuber V, Schultheiss N, Silver R, De Schutter E, Jaeger D (2011) Determinants of synaptic integration and heterogeneity in rebound firing explored with data-driven models of deep cerebellar nucleus cells. Journal of Computational Neuroscience 30:633–658

Stuart G, Spruston N, Sakmann B, Häusser M (1997) Action potential initiation and backpropagation in neurons of the mammalian CNS. Trends in Neurosciences 20(3):125–131

Stuart GJ, Sakmann B (1994) Active propagation of somatic action potentials into neocortical pyramidal cell dendrites. Nature 367(6458):69–72

Thomson AM, Lamy C (2007) Functional maps of neocortical local circuitry. Frontiers in Neuroscience 1(1)

Traub RD, Contreras D, Cunningham MO, Murray H, LeBeau FE, Roopun A, Bibbig A, Wilent WB, Higley MJ, Whittington MA (2005) Single-column thalamocortical network model exhibiting gamma oscillations, sleep spindles, and epileptogenic bursts. Journal of Neurophysiology 93(4):2194–232

Tsodyks MV, Markram H (1997) The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. Proceedings of the National Academy of Sciences USA pp 719–723

Tyrrell T, Willshaw D (1992) Cerebellar cortex: Its simulation and the relevance of Marr's theory. Philosophical Transactions of the Royal Society of London Series B: Biological Sciences 336(1277):239–257

Vervaeke K, Lőrincz A, Gleeson P, Farinella M, Nusser Z, Silver RA (2010) Rapid desynchronization of an electrically coupled interneuron network with sparse excitatory synaptic input. Neuron 67(3):435–51

Vetter P, Roth A, Häusser M (2001) Propagation of action potentials in dendrites depends on dendritic morphology. Journal of Neurophysiology 85(2):926–37

Vogels TP, Abbott LF (2005) Signal propagation and logic gating in networks of integrate-and-fire neurons. The Journal of Neuroscience 25(46):10,786–10,795

Vos BP, Maex R, Volny-Luraghi A, Schutter ED (1999) Parallel fibers synchronize spontaneous activity in cerebellar Golgi cells. The Journal of Neuroscience 19(11):RC6

Watts DJ, Strogatz SH (1998) Collective dynamics of "small-world" networks. Nature 393(6684):440–2

Wils S, De Schutter E (2009) STEPS: Modeling and simulating complex reaction-diffusion systems with Python. Frontiers in Neuroinformatics 3:15

Wolpert DM, Miall RC, Kawato M (1998) Internal models in the cerebellum. Trends in Cognitive Sciences 2(9):338–347

Yoshimura Y, Callaway EM (2005) Fine-scale specificity of cortical networks depends on inhibitory cell type and connectivity. Nature Neuroscience 8(11):1552–9

Zeeuw CID, Hoebeek FE, Schonewille M (2008) Causes and consequences of oscillations in the cerebellar cortex. Neuron 58(5):655–658

Zubler F, Douglas R (2009) A framework for modeling the growth and development of neurons and networks. Frontiers in Computational Neuroscience 3(0)